THESIS

CONFIGURATION AND MANAGEMENT OF WIRELESS SENSOR NETWORKS

by

Min Young Kim

December 2005

Thesis Advisor:	Gurminder Singh
Second Reader:	Arijit Das

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE: Configuration and Management of WIRELESS SENSOR NETWORKS		**5. FUNDING NUMBERS**
6. AUTHOR(S) Min Yung Kim		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public released; distribution is unlimited		**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Wireless sensor networks (WSNs) are expected to play an essential role in the upcoming age of pervasive computing. As a new research area, there are several open problems that need to be investigated. One such problem is configuration and management of WSNs. To deploy sensors efficiently in a wide area, we need to consider coverage, purpose and geographic situation. By considering these elements, we can make general deployment strategies. Another issue is management of various sensors in wide area. To handle these issues, we need approaches from different view, management levels, WSN functionalities, and management functional areas.

In this thesis, I describe some of the key configuration and management problems in WSNs. Then, I present a newly developed application to address these problems.

14. SUBJECT TERMS Wireless sensor networks, Deployment strategies, management Dimensions, Mote, Coverage			**15. NUMBER OF PAGES** 149
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

CONFIGURATION AND MANAGEMENT OF WIRELESS SENSOR NETWORKS

Min Y. Kim
Major, Korean Army
B.S., Korean Military Academy, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2005

Author: Min Y. Kim

Approved by: Gurminder Singh
Thesis Advisor

Arijit Das
Thesis Co-Advisor

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Wireless sensor networks (WSNs) are expected to play an essential role in the upcoming age of pervasive computing. As a new research area, there are several open problems that need to be investigated. One such problem is configuration and management of WSNs. To deploy sensors efficiently in a wide area, we need to consider coverage, purpose and geographic situation. By considering these elements, we can make general deployment strategies. Another issue is management of various sensors in wide area. To handle these issues, we need approaches from different view, management levels, WSN functionalities, and management functional areas.

In this thesis, I describe some of the key configuration and management problems in WSNs. Then, I present a newly developed application to address these problems.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS AND ACRONYMS

CCD	Charge Coupled Device
DoS	Denial of Service
LLA	Logical Layered Architecture
MIB	Mote Interface Boards
MPR	Mote Processor Radio
MSP	Mote Security Package
PIR	Passive Infrared
QoS	Quality of Service
RF	Radio Frequency
UTM	Universal Transverse Mercator
VFA	Virtual Force Algorithm
WSNs	Wireless Sensor Networks

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This thesis is dedicated to my family. My lovely wife, Sook, and our children, Myoung-Chul and Myoung-Hyuen, have constantly supported me with their love, encouragement, and patient.

I would also like to acknowledge my advisors, Gurminder Singh and Arijit Das, for their continual guidance, mentorship, and support throughout this thesis. They have patiently encouraged my efforts with their inspirational comments. Without their help, this work would never reach the level of quality that I have always wanted in my life.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Wireless sensor networks (WSNs) are expected to play an essential role in the upcoming age of pervasive computing. As a new research area, there are several open problems that need to be investigated. One such problem is configuration and management of WSNs.

When we deploy a sensor network, although there are suggested configurations to determine sensor placement for a given target distribution, we need to check whether this deployment will be effective for the given terrain. To check the state of the WSNs, we need tools which police the state and identify nodes failure.

The task of developing management systems is very complex and has several constraints and considerations. For instance, it needs to manage tens of thousands of network elements with particular features and organization. The task becomes worse when we consider the physical restrictions of the sensor nodes, in particular energy and bandwidth restrictions.

B. OBJECTIVES

The first object of this thesis is research for configuration and management of WSNs. The second object is developing configuration and management application of WSNs.

C. RESEARCH QUESTIONS

The primary target of this thesis is a configuration and management of WSNs. The study will be in accordance with the following questions.

What are the basic characteristics of Sensor Network?

- Characteristics of motes.
- What is the limit of WSNs?
- Overview and evaluation of existing implementations
- How to configure WSNs?
- What are the restrictions in configuration of WSNs?
- What are the existing deployment methods?
- How to deploy the sensor node?
- How to manage WSNs?
- What are the constraints in management of WSNs?
- What are the existing management methods?
- How can we identify and report failures in sensor networks?

D. SCOPE

The scope of this research can be divided into two parts. The first one will be research on configuration and management of WSNs. In the second part, I will implement interface application which will be used for that purpose using Crossbow's hardware technology.

I will focus my research on the limitation in configuration and management of WSNs. Then I will research the existing methods for configuration and management of WSNs. My interface application will work for deployment in designing stage and tell the status of each node in managing stage.

E. THESIS ORGANIZATION

This thesis is organized as follows. Chapter II describes configuration challenges and deployment strategies for WSNs. Chapter III covers management challenges and dimensions for WSNs. Chapter IV describes characteristics of Crossbow’s MSP410CA security system which was tested for this thesis. Chapter V covers configuration and management application of WSN which was developed by us and describes the application’s requirement and component. Chapter VI describes summary, conclusion and future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. SENSOR NETWORK CONFIGURATION

A. INTRODUCTION

A WSN consists of a large number of sensor nodes deployed over an area and integrated to collaborate through a wireless network. In this meaning, how to deploy a sensor node is fundamental problem in WSN. It has a great impact on efficiency and cost. So when we deploy a sensor node, we need to think about consideration like purpose of deployment and the topography and coverage ability of a sensor node. Most of all the purpose of WSN is to get large coverage with as few sensor nodes as possible.

As an example, consider the millions of acres that are lost around the world, due to forest fires every year. In all fires, early warnings are critical in preventing small harmless brush fires from becoming monstrous infernos. By deploying specialized wireless sensor nodes in strategically selected high-risk areas, the detection time for such disasters can be drastically reduced, increasing the likelihood of success in early extinguishing efforts. Also, since the nodes are self-configuring and do not need constant monitoring, the cost of such a deployment is minimal compared to the huge losses incurred in a large blaze (Meguerdichian, Koushanfar, Potkonjak & Srivastava, 2001).

In this Chapter, I will speculate about problems in configuration of WSNs followed by discussion of existing strategies to deploy sensor nodes usefully.

B. CONFIGURATION CHALLENGES

Configuration involves procedures related to planning, placement, and self-organization of a WSN. The configuration (predeployment) is related to the:

- Definition of WSN application requirements
- Determination of the monitoring area (shape and dimension)
- Characteristics of the environment
- Choice of nodes
- Definition of the WSN type
- Service provided

In the deployment phase, sensor nodes can be placed by dropping them from a plane, rocket, or missile, and placed one by one by a human or a robot. Any placement approach for sensor nodes must also take into account the expense and difficulty in redeploying nodes. This is chiefly due to the limited life span of nodes and to their generally no replaceable power sources (Mehrotra, 2001). Another problem is the optimal location of the access point (sink node or base station). An inefficient configuration may adversely affect overall performance. WSNs are application specific, which means that the configuration changes from one WSN to another (Ruiz et al., 2005).

Mostly, WSNs are deployed to monitor certain areas or object, for example, to detect intruders, wildfires, or rare animals in a habitat. To detect objects, sensors are needed to be in a close enough range to sense the object or event. Now, we have two questions. One is which points or places are close to enough to sensor such that an event taking place at this point can be sensed. In other words, which points are covered? Coverage is thus an important aspect of QoS in sensor network.

The second question is: Given an area to be observed and some coverage requirement, what number of sensors is needed and where should they be placed? This question can be posed under several interesting constraints, for example, cost constraints, presence of obstacles, availability of different types of sensors, and so forth (Karl & Willig, 2005).

In WSNs, coverage has a two-fold meaning: range and spatial localization. Range refers to the geometric area of a designated sensing mission, while spatial localization emphasizes the relative spatial positions of sensor nodes and targets so as to extract accurate measurement (Wang, Hassanein & Xu, 2005).

Meguerdichian and colleagues define the coverage problem from several points of view including deterministic, statistical, worst and best case, and present examples in each domain (Meguerdichian et al., 2001).

Meguerdichian et al. define the attempts which are made to quantify the quality of service by finding areas of lower observability from sensor nodes and detecting breach regions in worst-case coverage. On the contrary, finding areas of high observability from sensors and identifying the best support and guidance regions are of primary concern in best-case coverage. They also propose optimal polynomial-time algorithms for solving each case (Meguerdichian et al., 2001).

C. SENSOR CONFIGURATION STRATEGIES

1. General Strategies

Sensor configuration strategies may vary depending on purpose, cost and application. Even though, a lot of deployment strategies can be chosen for WSNs, Wang et al. (2005) generalize four methods of sensor deployment. The following section will detail these strategies.

a. Predetermined Strategy

Two situations are considered in predetermined strategy. In first situation, the environment or target is specified. So we can deploy sensor node by this information or specified target. In second situation, we can deploy sensor node at regular intervals. In this situation, we need some grid-based topology in which the sensing site is spatially modeled as a grid-based distribution. Dhillon, Chakrabarty and Iyengar (2002) determined the granularity of the grid (distance between adjunctive grid points) by the desired accuracy. Wang et al. (2005) noted that predetermined strategy can provide an optimal solution for desirable coverage and obtain high Qos and cost efficiency at the same time. But the problem of the first situation is that we often can not obtain knowledge of the environment and target. On the contrary, a regular grid-based approach has better adaptation to the variation of the conditions. But, it also has some drawbacks. The grid coverage relies on accurate sensor detection in reality, although sensor detection is often uncertain.

b. Self-regulated Strategy

Self-regulated strategy is developed to overcome the difficulties of the predetermined approach. Howard, Mataric and Sukhatme (2002) propose a potential field-based method to deploy sensor nodes automatically in an unknown environment. Because the sensing fields are established in a manner in which each sensor node is repelled by obstacles and by other nodes, the entire network is self-spread throughout the environment and can reach the maximum coverage. Clouqueur et al. (2002) present a scheme to deploy sensor nodes sequentially in steps by introducing path exposure as a metric of goodness. With the strategy of properly choosing the number of sensors in each step, the cost of deployment can be minimized to achieve the desired detection performance. Self-regulated methods are scalable to increasing the number of sensor nodes, but the computational expense may become prohibitive.

c. Randomly Undermined Strategy

Randomly undermined strategy is more realistic for a large-scale WSN application, such as unknown battlefields or hostile terrains. With this approach, sensor nodes are generally spread uniformly in a given area (Tilak, Abu-Ghazaleh & Heinzelman, 2002). This strategy is preferable because of easy placement of nodes and therefore low cost. Although sensing devices can be randomly deployed in two- or three-dimensional spaces, the coverage might not be uniform due to obstacles or other sources of noise in the environment. Based on an initial random distribution, Zou and Chakrabarty (2003) introduced a practical virtual force algorithm (VFA) to reposition the sensors in order to enlarge coverage to the desired optimal results, thus dealing with cases of high- and low-detection accuracy while considering energy constraints.

d. Biased Distribution Strategy

The uniform deployment of sensor nodes may not always satisfy the design requirements but biased deployment may. Willig et al. (2002) illustrate an example of biased placement of sensors in a large-scale office building in which the density of sensor nodes close to the windows is much higher than that in the middle of the room. Some comparisons of different deployment strategies by means of simulations have been presented by Tilak et al. (2002).

Ray et al. (2003) proposed a new framework. In their framework, they provide robust location detection in harsh environment by overlapping sensor coverage. A unique set of sensors cover each position in region. To reduce the number of active sensors required by the system, they use identifying code theory, which keep a large number of sensors to be maintained in an energy-saving mode. They have also proposed a polynomial-time algorithm ID-CODE for determining sensor placement by generating a corresponding irreducible identifying code.

2. Energy Efficient Strategies

Irali, Maleki and Predram (2005) investigated and developed energy-efficient strategies for deployment of WSNs (WSN) for the purpose of monitoring some phenomenon of interest in a coverage region. They first described a two-level WSN structure where the sensors in the lower level monitor their surrounding environment and the micro-servers in the top level provided connectivity between the sensors and a base station. Irali et al. then formulated and solved the problem of assigning positions and initial energy levels to the micro-servers and concurrently partitioning the sensors into clusters assigned to individual micro-servers so as maximize the monitoring lifetime of the two-level WSN subject to a total energy budget. This problem, called MDEA, was solved for both collinear deployment and planar deployment situations.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SENSOR NETWORK MANAGEMENT

A. INTRODUCTION

As both the environment of a WSN and the WSN itself change, the system has to adapt. It has to monitor its own health and status to change operational parameters or to choose different trade-offs. In this sense, the network has to maintain itself; it could also be able to interact with external maintenance mechanisms to ensure its extended operation at a required quality (Mainwaring et al., 2002)

Until now, WSNs and their applications have been developed without considering an integrated management solution. The task of building and deploying management systems in environments that will contain tens of thousands of network elements with particular features and organization and that deal with the aforementioned attributes is not trivial. This task becomes more complex due to the physical restrictions of the unattended sensor nodes, in particular energy and bandwidth restrictions. (Ruiz, Nogueria & Loureiro, 2005)

In this chapter, my focus is on medium-sized WSN management. Clearly, the mechanisms associated with traditional management paradigms must be rethought. In this sense, a new paradigm called autonomic management is explored. The rest of this chapter is organized as follows. At first, I will present an overview of network management and discuss the management challenges for WSNs. Then, management dimensions (management levels, WSN functionalities, and management functional areas) are presented.

B. MANAGEMENT CHALLENGES

In this section, we will compare the management of traditional networks with the management of WSNs and discuss important characteristics of WSNs that make their management different from that of other networks.

In planning stage, the traditional computer networks are designed to assist a variety of applications. The network tends to follow established planning of available resources and the location of each network element is renowned. In a WSN, the network is planned to have unattended operation. For instance, the initial configuration of a WSN can be quite different from what was supposed to be in cases such as deploying the nodes to a sea, forest, or other remote regions. Another difference is topology status. In WSN, topology is dynamic because sensor nodes can become out of service temporarily or permanently (nodes can be discarded, lost, destroyed, or even run out of energy).

An essential characteristic of managed WSNs based on an autonomic system, which is a method to self-managed computing systems with a minimum of human interference. WSNs management requires capability of self-configuration, self-organization, self-healing, and self-optimization. System configuration must arise automatically. It must be able to recover from routine and unexpected events that might cause some of its parts to malfunction. The network must be able to discover problems or potential problems, such as uncovered area, and then find an alternate way of using resources or reconfiguring the system to keep it functioning smoothly. A managed WSN also looks for ways to optimize its functioning. A managed WSN needs to know its environment and the context surrounding its activity and act accordingly (Ruiz et al., 2003).

The task of building and deploying autonomic management systems with tens of thousands of network is very complex. This task becomes even more involved due to the physical restrictions of the sensor nodes, in particular energy and bandwidth restrictions. The management application to be built also depends on the kind of application being monitored. To deal with complex management situations, using management dimensions is a good strategy.

C. MANAGEMENT DIMENSIONS

The following section involves how the traditional management dimensions can be valid in WSN management. Also, new dimension for WSN management (Ruiz et al., 2005) is proposed that reflects the general features of the different types of the networks.

1. Dimensions for WSN Management

The management application needs to be compatible with the monitored application because WSNs are embedded in applications to monitor the environment and act upon it. It is needed to distinguish the WSN and establish a novel management dimension to develop WSN management services and functions. Thus, Ruiz et al. (2005) categorizes five main WSN functionalities: configuration; sensing; processing; communication; and maintenance. A novel dimension for the management is defined by these functionalities, as presented in Figure 1. In this way, abstractions offered by management functional areas, management levels, and WSN functionalities construct an organization of WSN management. The novel dimension introduced can be observed in the upper part of Figure 1.

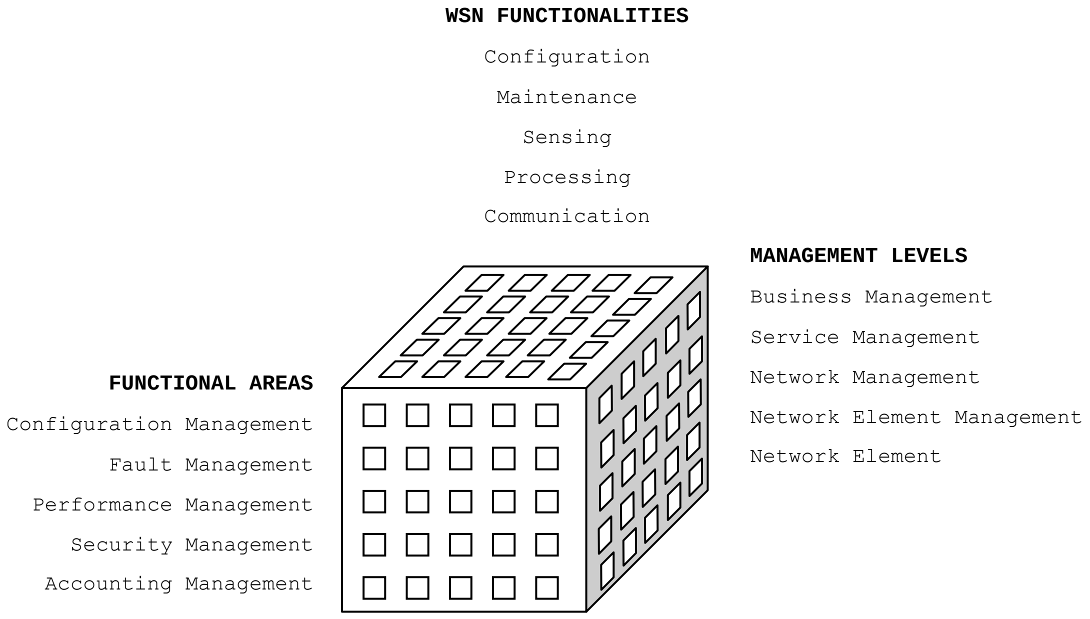


Figure 1. Management dimensions for WSNs (From : Ruiz et al, 2003)

The next subsections describe WSN management from the angle of management level, WSN functionalities, and management functional areas.

2. WSN Functionalities

Ruiz et al. (2003) proposed the novel dimension for the WSN functionalities composed by the configuration, sensing, processing, communication, and maintenance functionalities. These WSN functionalities can be observed in the upper part of Figure 1. This novel dimension presents a scheme to characterize WSNs considering that they are application dependent.

a. Configuration

Configuration is connected with planning, placement, and self-organization of a WSN. Ruiz et al. (2003) discussed the configuration considering the possible types of WSN and the other two management dimensions.

WSNs can be classified in various ways when we consider the network management level and management functional areas based on configuration functionality. Various kinds of WSNs and key elements are described in below.

- Homogeneous/ heterogeneous: same hard ware
- Hierarchical/ flat: nodes are grouped for the purpose of communication
- Stationary/ dynamic: WSN is static
- Symmetric/ asymmetric: each transceiver has the same transmission range
- Regular/ irregular: placement of node
- Sparse/ dense: number of nodes per area unit

b. Sensing

The autonomous sensor nodes provide the lowest level of the sensing application. Data gathering is an important operation in a sensor network. Sensing functionality depends on the kind of the phenomenon. Thus, WSNs can be organized to continuous, reactive and periodic. In case of coexisting approaches in the same network, this model is referred to as the hybrid collect model. Temperature detection is an example of a continuous phenomenon and a sensor deployed for animal detection is an example of an application in which the phenomenon is moving.

Other examples of phenomena are video; audio; pressure; mechanical stress; humidity; soil composition; luminosity; seismic; and chemical.

Whether gathering is continuous or not, WSNs are defined depending on how the data will be transmitted to the observer. The sensing includes the exposure (time, distance, and angle of phenomenon exhibition at the sensor), calibration, and sensing coverage. It will be wasteful if all sensor nodes are active all the time depending on the density of the phenomenon. A model that is compatible to this case is the Frisbee model (Cerpa et al., 2001). On the other hand, redundancy (overlapping in the sensor coverage) should be utilized in such a way that fault tolerance in the communication network is avoided and better accuracy can be found in Vieira et al. (2003). In any case, the sensors can be mobile. In this case, the sensors are moving with respect to each other and to the observer as well, and they have direction, orientation, and acceleration.

c. Processing

Memory and processor of a sensor node form the computational module, which is a programmable part that offers computation and storage for other nodes in the system. Algorithms must be developed depending on the communication constraints of the system. The computational module performs basic signal processing (e.g., simple translations based on calibrating data or threshold filters) and dispatches the data according to the application. Processing can also involve correlation procedures such as data fusion, which merges one or more data packets received from various sensors to create a single packet (data fusion). Data fusion allows design of a network that delivers required data while meeting energy requirements and helps to reduce the amount of data transmitted between the sensor nodes and the observer.

d. Communication

Individual nodes communicate and coordinate among themselves. Infrastructure and application are proposed for two types of communication. The communication needed to configure, maintain, and optimize operation is referred to infrastructure communication. The configuration and topology of the sensor network may be varying depending on the presence of a hostile environment and nodes that fail regularly. In this case, new protocols are required to promote WSN productivity. In a static sensor network, an initial stage of the infrastructure communication is required to establish the network and an additional communication is needed to complete its reconfiguration. If the sensors are mobile, additional communication is needed for path discovery/reconfiguration.

Application communication relates to the passing on sensed data (or information obtained from it). Transmitting and receiving a data packet needs fixed cost energy related to the hardware and a variable cost that depends on the distance of transmission. Therefore, short distance transmissions are preferred to conserve energy. In a homogeneous and flat WSN, the sensor nodes can establish a multi-hop network by forwarding each other’s messages, which can provide different connectivity options. In a heterogeneous and hierarchical WSN, the cluster heads can establish a single-hop network for reporting aggregated data to the BS. Within a cluster, measured data are sent to the cluster head by the sensor nodes under its control. All nodes in a cluster are the same except in the heterogeneous WSN, where the cluster head has a larger transmission capacity.

Considering the data delivery required by the application interest, WSNs can be classified as continuous, when sensor nodes accumulate data and send them to an observer continuously along the time, and as on demand, when they answer an observer’s query. A WSN is event driven when sensor nodes send data referring to events happening in the environment and programmed when nodes accumulate data according to conditions defined by the application. The hybrid model is used for mixed approaches in the same network. The cost of sending data continuously may lead to a more rapid consumption of the limited network resources and, thus, shorten resource lifetime.

Multi-hop wireless capabilities will allow communication and coordination among autonomous nodes in unexpected environments and configurations. At the same time, wireless channels present challenges of dynamic operating conditions, power constraints for autonomously-powered nodes, and complicating interactions between high level behavior and lower level channel characteristics.

The communication approach can be organized as below for any of the above models:

- Flooding (sensors broadcasting their information to their neighbors, which in turn broadcast these data until they reach the observer)
- Gossiping (sending data to one randomly selected neighbor)
- Bargaining (sending data to sensor nodes only if they are interested)

- Uni-cast (sensor communicating to the sink node, cluster head, or BS directly)
- Multicast (sensors forming application-directed groups and using multicast to communicate among group members)

e. Maintenance

The WSNs uses maintenance functionality to enable configures, protect, optimize and heal themselves without a lot of input from the human operators. Maintenance notices failures or performance degradations, initiates diagnostic procedures, and carries out corrective actions on the network. Its ability to detect changes in the network state allows the self-management to adapt and optimize the network behavior. Beyond corrective maintenance, the other types of maintenance are adaptive, preventive and proactive. The system should adapt to meet the changes in adaptive type, learn to anticipate the impact of those changes in preventive type or learn to intervene so as to preempt negative events in proactive type. An example of maintenance concerns the density of nodes in the WSN. For instance, in case of a high node density, the maintenance can turn off some nodes temporally.

The WSN state (e.g., topology, energy, coverage area) changes regularly. In the case of static networks, changes happen because nodes may become unavailable during operation. This dynamic behavior must be observed. The maintenance depends on the knowledge of the network state. Thus, maintenance functionality is needed to keep the network operational and functional to guarantee robust operation in dynamic environments, as well as optimize overall performance. Maintenance provides dependability, the main attributes of which are reliability; availability; safety; security; testability; and perform ability. WSNs have important characteristics based on the application. Some of them are:

- Planning
- Deployment
- Coverage
- Accuracy
- Fidelity
- Density
- Self-organization
- Adaptation
- Location

3. Management Levels

Different management levels are discussed in this subsection. Management levels consist of business, service, network, network element management and network element.

a. Business Management

Business management manages development and determination of cost functions because WSNs depend on applications. A sensor networks can be represented as a cost function associated with network setup, sensing, processing, communication, and maintenance. WSN applications have gigantic possible profits for society as a whole and represent new business opportunities.

b. Service Management

WSN services are related with functionalities associated with application objectives. Basic WSN services are sensing, processing, and data dissemination. Two main issues, quality of service (QoS) and denial of service (DoS) are associated with WSN service management:

(1) Quality of service. To be effective and provide guaranteed services, Qos architectures require QoS elements to be adequately configured and monitored. So management applications must manage QoS aspects. QoS models, QoS sensing, processing, and QoS dissemination are components involved in QoS support to WSNs. If the number of monitored QoS parameters is large, the energy consumption will be increased and the network lifetime will be reduced.

(2) QoS model. A QoS model identifies an architecture in which some of the services can be presented in WSNs. All other QoS components, such as QoS sensing, QoS processing, and QoS dissemination (e.g., signaling, QoS routing, and QoS MAC), must cooperate to accomplish this goal. A management application can set up the QoS model and manage the QoS signaling that coordinates behavior of the other components. QoS-related tasks must be performed by using network management functions.

(3) QoS sensing. QoS sensing considers the sensor device calibration, environment interference monitoring, and exposure (time, distance, and angle between sensor device and phenomenon). Meguerdichian et al. (2001) defines coverage area as a measure of QoS for a WSN. In the worst-case coverage, attempts are made to quantify the quality of service by finding areas of low observability to sensor nodes and detecting breach regions. In the best-case coverage, the management application must find areas of high observability to sensors and identify the highest accuracy. A denser network will lead to more effective sensing because of the higher accuracy of the network and better fault

tolerance. But, this will cause to collisions, congestion situations, increase latency and reduce energy efficiency. Thus, the management application requires the flexibility of meeting the performance demands by controlling the reporting rate of sensors, controlling the virtual topology of the network (by turning off some sensors), or optimizing the collective reduction communication operation (by data aggregation).

(4) QoS dissemination. Energy and dynamic topology of WSNs make QoS dissemination a challenging task. The two components for QoS dissemination are QoS routing and QoS medium access control (MAC). QoS routing discovers a path that meets with a given QoS requirement, and QoS MAC solves the problem of medium contention that supports reliable uni-cast communication (Wu and Harms, 2001).

(5) QoS processing. Processing quality is decided by not only processor and memory capacities buy by the robustness and complexity of the algorithms used. The computing paradigm is more dependent on one driven by data than one based on computational power. The way to measure processing performance is more dependent on the immediacy and accuracy of the response and energy consumption than processor speed. Individual computers become less important than lower granularity and dispersed computing attributes.

The energy consumption to execute a service with a determined quality level decides the network quality of service. In most WSNs, energy consumption is one of the main concerns. However, in urgent situations, the network needs to apply the maximum of energy possible to deliver information — for instance, in WSNs deployed over the forest to monitor a fire as much information as possible is needed in the shortest time period. In this case, proper and fast delivery of information is more important than the network lifetime

Any situation that reduces the capacity of the network to perform its expected job is called DoS (denial of service). Some examples of incidental threats are hardware failures, software bugs, resource exhaustion, and unexpected environmental conditions. DoS aspects will be discussed in Subsection Security Management.

c. Network Management

This layer aims to deal with a network, especially when the network is distributed over a wide geographical area. Relationships among sensor nodes are to be considered in the network management level. Individual nodes need to sense, process data, and communicate, thus contributing to a common objective. In this way, nodes can be involved in collaboration, connectivity, and aggregation relationships.

d. Network Element Management

Managed network elements characterize the sensor and actuators nodes or other WSN entities, which perform management functions and present sensing, processing, and dissemination services. The basic functions of a WSN management network element are

- Power management (how a sensor node uses its power)
- Mobility management (how the movement of sensor nodes is planned, run, and registered)
- State management (how a sensor node manages the three management states defined for a node: operational, administrative, and usage)

- Task management (how a sensor node balances and schedules the sensing, processing, and dissemination tasks given to a specific network state)

To perform coordinated activities with global objectives, each sensor node needs to be autonomous and capable of organizing itself in the overall community of sensor nodes.

Considering processing power, memory capacity, battery lifetime, and communication throughput, sensor nodes have strong hardware and software restrictions. These are typical characteristics of mobile and wireless. Thus, software designed for a sensor node must consider these limitations. The most concerned physical restriction of a WSN is the obtainable energy because batteries are often not revived during the operation of a sensor node and all activities performed by the node must take energy consumption into account.

e. Network Element

The network element is composed of physical and logical components of a managed element. Physical resources involve sensor or actuator nodes; power supply; processor; memory; sensor device; and transceiver. Logical resources involve communication protocols; application programs; correlation procedures; and network services. Because applications may require networks with a large number of sensor nodes, a network element can deal with a single node component or a group of nodes. A manageable element can be a cluster of nodes or a cluster-head node, rather than an individual node in such a case.

Understanding node capability allows function management to be structured and fine-tuned more efficiently. The physical aspects of a network element are explained in the following.

(1) Power Supply. The most widely used power supply in a WSN is the battery, which is classified into the following types (Savvides et al., 2001):

- Linear model — the battery is regarded as a bucket of energy that is linearly drawn from this bucket by the energy consumers
- Dependent model — takes into account the rate at which energy is drawn from the battery to compute the remaining battery lifetime; at high discharge rates, the capacity of the battery is reduced
- Relaxation model — considers a phenomenon seen in real-life batteries in which the battery's voltage recovers if the discharge rate is decreased

(2) Computational Module. This module is composed of processor and memory. It is responsible for the collaborative processing between nodes to achieve the levels of service and reliability desired by the observer.

(3) Sensor element. Sensing devices can be classified into three groups: monitors (e.g., magnetometer, light sensor, temperature, pressure, humidity); motion detectors (e.g., accelerometer); and media processing (e.g., audio, video).

(4) Transceiver Radio frequency (RF), infrared, and optical are considered to the main types of a transceiver. RF communication is based on electromagnetic waves with frequencies ranging from tens of kilohertz to hundreds of gigahertz. The size of the antenna is the most important factors in the design of RF communications. An antenna should be at least l/4 to optimize transmission and

reception, where l is the wavelength of the carrier frequency. A transmitting device uses a laser beam to send information in optical laser communication. An optical receiver receives the signal and decodes the data in the form of a photodiode or charge-coupled device (CCD) array. Optical communications can be categorized into two types: passive (the laser signal is generated through a secondary source) and active (the transmitting device generates its own laser signal).

(5) Software. This represents programs and procedures that becomes an autonomous system capable of performing the information processing, relaying, or routing.

4. Management Functional Areas

WSN management takes into account fault, security, performance, and accounting management functional areas extremely dependent on the configuration functional area. In WSNs, all operational, administrative, and maintenance characteristics of the network elements; the network, services; and business; and the adequate execution in the activities of configuration, sensing, processing, communication, and maintenance are dependent on the configuration of the WSN. This idea is depicted in Figure 2, in which the configuration functional area plays a central role.

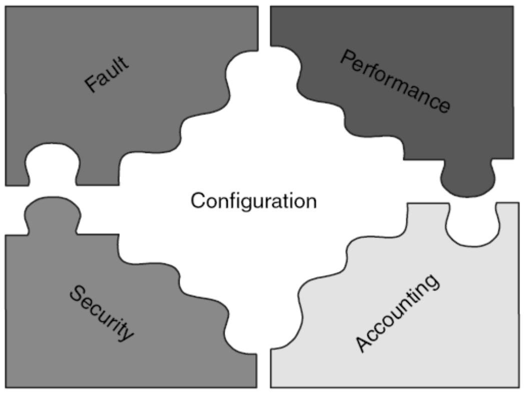


Figure 2. The role of configuration management (From: Ruiz et al, 2003)

a. ***Configuration Management***

Any problem or situation not anticipated in the configuration phase can affect the offered service because the objective of a sensor network is to monitor (acquisition, processing, and delivery of data) and, eventually, to control an environment. The configuration management must present self-organization, self-configuration, self-discovery, and self-optimization. Some management functions defined for network level configuration management are:

- Requirements specification of the network operational environment
- Monitoring of environmental variations
- Size and shape definition of the region to be monitored
- Node deployment — random or deterministic
- Operational network parameters determination
- Network state discovery
- Topology discovery
- Network connectivity discovery
- Control of node density
- Synchronization

- Network energy map evaluation
- Coverage area determination
- Integration with observer

Some management functions defined for network-element level configuration management are:

- Node programming
- Node self-test
- Node location
- Node operational state
- Node administrative state
- Node usage state
- Node energy level

b. Fault Management

Faults in WSNs are tend to occur frequently and not an exception. This is one of the reasons why management of WSNs is different from the traditional network management. Due to energy shortage, connectivity interruption, and environmental variations, faults happen all the time. In general, sensor networks need to be fault tolerant and robust and require surviving in spite of occurrences of faults in individual nodes, in the network, or even in services provided. Other events can happen in a WSN related to communication in addition to events caused by energy problems; quality of service; data processing; physical equipment fault; environment; integrity violation; operational violation; security; and time-domain violation. So, even if a node has an adequate energy level to perform its function, it may decide not to do that for other reasons.

Fault management must provide basic characteristics such as self-maintenance, self-healing, and self-protection.

In a WSN, Failures happen frequently, and fault management is a critical function. Several characteristics of sensor networks propose that faults, common in traditional computer networks, will be even more common in this kind of network. Fault management will play an equally, if not more, crucial role in WSNs.

c. Performance Management

In performance management, the higher the number of managed parameters, the higher the energy consumption and the lower the network lifetime are. On the other hand, if parameter values are not attained, it may not be possible to manage the network properly.

The configuration (in terms of sensor capabilities, number of sensors, density, node distribution, self organization, and data dissemination) is important in determining the performance of the network. Performance management must deal with the self-service characteristic. As such, the performance of the network and provided service are best measured, when it meets the accuracy and delay requirements of the observer, as well as consumed energy.

The accuracy specifies the reliability or exactness of a result. The accuracy of a measurement at a network element (sensor) is specific to the physical transducer and the nature of the phenomenon. At the network level, accuracy is determined by the delay in data delivery due to network congestion, route length, duty cycle of the sensors, or aggregation processing of data. Accuracy at the service level depends on the metric chosen by the application for establishing the coverage area and amount of energy to be spent in gathering and disseminating data. Because multiple samples will be received from different sensor nodes and with different data quality, additional performance metrics need to include:

- Coverage area
- Exposure
- Sensor cost
- Scalability
- Produced data quality

In some applications, in addition to information about some features of the phenomenon, it might be essential to know where (sensor location), when (data-time), and how (sensor calibration, exposure) to manage the WSN performance.

d. Security Management

Due to ad hoc organization, intermittent connectivity, wireless communication, and resource limitations, security functionalities for WSNs are complicated to provide. A WSN is threatened by different safety threats: internal, external, accidental, and malicious. Information or resources can be destroyed; information can be modified, stolen, removed, lost, or disclosed and service can be interrupted. Even if the WSN is secure, the environment can change it insecure or vulnerable. Security management must provide self-protection, reliability, disposability, privacy, authenticity, and integrity.

Determining if a fault or collection of faults is the result of an intentional DoS attack presents a concern of its own — a point that becomes even more difficult in large-scale deployments, which may have higher nominal failure rates of individual nodes than small networks will. The robustness against physical challenges may avoid some classes of DoS attacks. Each layer of the protocol stack is susceptible to various DoS attacks and has several options available for its defense.

e. Accounting Management

Accounting management involves functions connected to the use of resources and corresponding reports. It establishes metrics and quotes and limits what can be used by functions of other functional areas. These functions can trace the behavior of the network and even make inferences about the behavior of a given node. Accounting management must be considered self-sustaining.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. OVERVIEW OF THE TESTING HARDWARE AND SOFTWARE

A. INTRODUCTION

This chapter presents overview of the hardware and software which are used in this thesis. I will introduce MSP410CA Mote Security System, a Crossbow commercial product and software; MOTE-VIEW client software, XServe and Surge-view.

B. HARDWARE

1. Overview

MSP410CA Mote Security System consists of eight nodes which use battery power and target at serving security implementations. The MSP410 mote is the WSN component in configuration and management application that is responsible for creating and maintaining the wireless ad-hoc mesh network and then for collecting and returning to the base station the sensor's values that will be shown in the application. MBR410CA base station is interface between MSP 410 motes and computer. In the following sections the MSP410CA system will be explained and Crossbow implementations will also be proposed. Figure 3 shows elements of the kit.

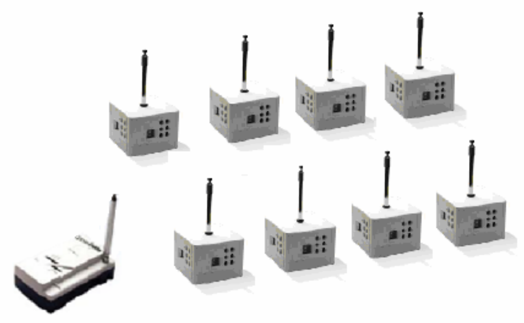


Figure 3. Mote Security System MSP 410CA

2. Proposed Deployments

A variety of security applications can be developed by the MSP410CA Mote Security System support. MSP410 Motes can be deployed in a perimeter or grid pattern in a typical security application. The MSP410 Mote is able to generate detection by transmitting the proper sensor's data to the base station directly or through the network and by combining wireless mesh networking technology and carrying a set of sensors. Figure 4 presents a possible perimeter deployment around a building and shows the distances between the motes which recommended by Crossbow. Figure 4 also shows the orientation restriction recommended by Crossbow, to achieve better results. In the MSP410 Sensing Subsystem section we will further study this constraint.

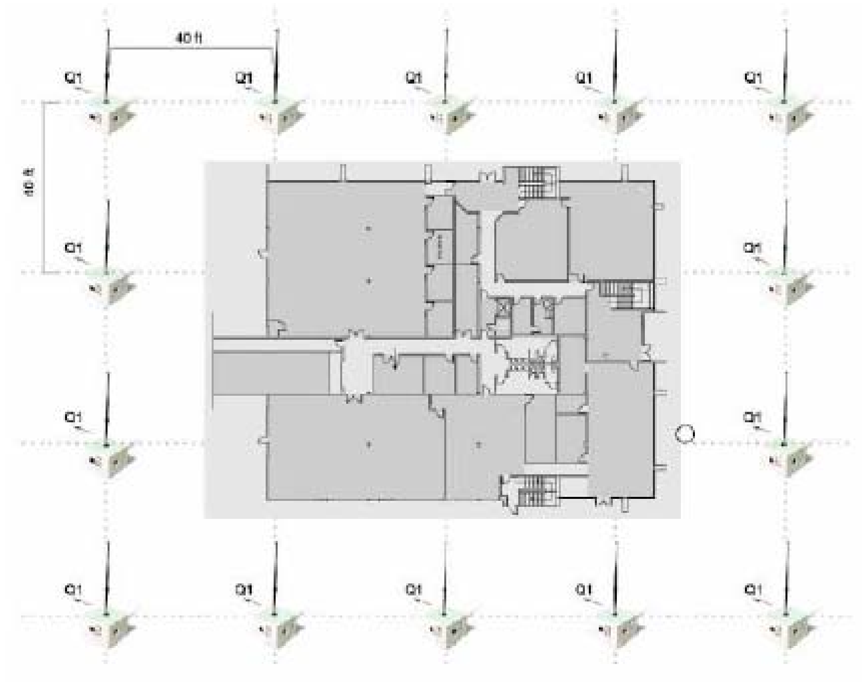


Figure 4. MSP410 Mote deployment for perimeter monitoring: MSP410 Series User's Manual (from: Crossbow, 2005)

Figure 5 shows the proposed dense grid deployment presented in the user's manual. The proposed deployment contains the suggested distances and the same orientation restriction with the perimeter option. The purpose of the above dense grid is to provide complete coverage of the area of interest. It is not the communication range but the average sensor's effective distances restricts the distances in both proposed. Greater distances can be used between the motes, in case the application's requirements do not indicate complete area coverage. Thus a greater area will be covered by the same number of motes, but possibly, some shadow areas also will be formed.

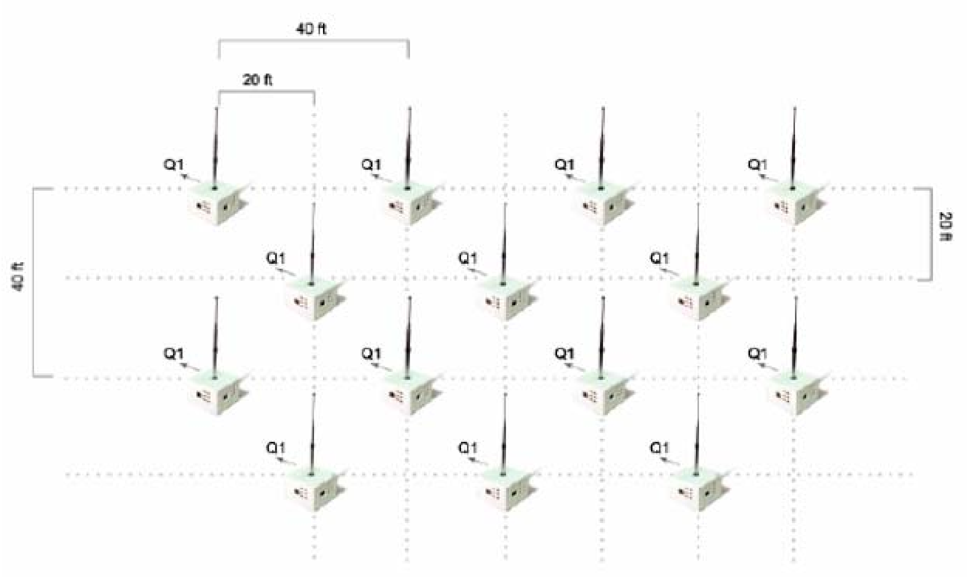


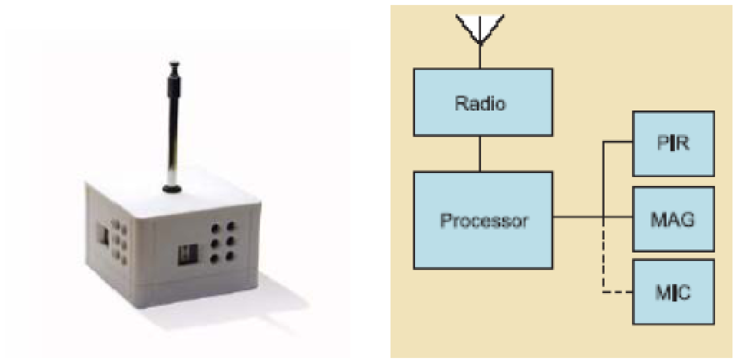
Figure 5. MSP410 Mote deployment for a dense grid monitoring: MSP410 Series User's Manual (Crossbow, 2005)

3. Systems Components

The MSP410 Mote Security System consists of two parts. The first part, MSP410 Motes is responsible for the sensing functions and for the deployment and maintenance of the wireless mesh ad-hoc network. The other part MBR410CA, the base station, acts as the important WSN interface with other systems. The base station is responsible for delivering the collected data to the connected system. The base station is also used to reprogram the motes. The following sections detail further the above system's components, based on the information included in the MSP410 Series User's Manual and the MPR/MIB User's Manual (Crossbow, 2005).

4. MSP410 Mote MICA2 Platform Core (Microcontroller, Radio)

The MICA2 processor/radio board and a variety of sensors comprise the mote, the core element of the MSP410CA system. Figure 6 shows the mote in the "heat reflective plastic enclosure" and the mote's basic block diagram. This section focuses on the platform, microcontroller, and radio.

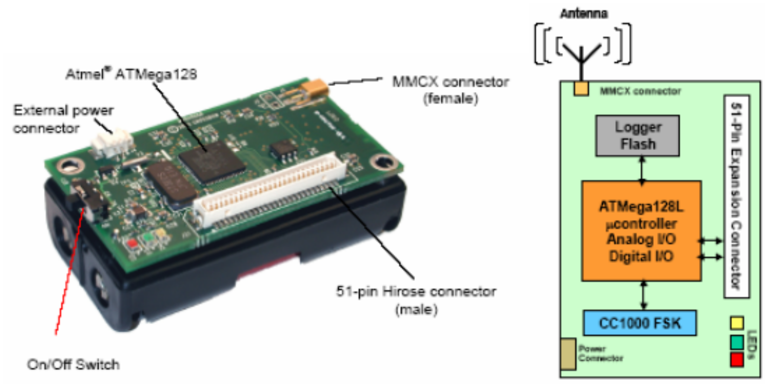


(a) (b)

Figure 6. (a) MSP410 Mote and (b) Mote's basic block diagram MSP410_Datasheet (http://www.xbow.com)

The processor/radio part, the left part of the above block diagram, belongs in the MICA2 Crossbow products' family. The MPR/MIB User's Manual separates MICA2 into three models based on their RF frequency band: the MPR400 (915

MHz), the MPR410 (433 MHz) and the MPR420 (315 MHz). The MSP410CA system uses the second model, the MPR410. All the MICA2 models are compatible and can communicate with each other. Figure 7 shows the platform and the block diagram of the MICA2.



(a) (b)

Figure 7. (a) Photo of a MICA2 (MPR4x0) without antenna, (b) MICA2 block diagram of a MPR/MIB User's Manual (Crossbow, 2005)

The Amtel Atmega128 microcontroller is the basic platform element which has total control of the functions. The external flash and the 64-bit Serial ID number are two peripherals which are directly connected to the processor and all the sensors and devices which are not directly connected to the processor are also handled as peripherals. The 51-pin Hirose interface connector provides wired communication and reprogramming functions

The other important element of the MICA2 Platform Core is the Chipcon CC1000 radio. It manages transmission at an effective baud rate 19.2 kilobit per second (kbps). Two-tone

Frequency Shift Keying (FSK) modulation and Manchester encoding is used for transmission (MSP410 Series User's Manual [Crossbow, 2005]). Within the specified band, the radio can be tuned up to four different channels around 433MHz,. The actual number of possible channels is higher, but the recommended channel spacing in order to avoid interference is greater than 500 kHz.

5. MSP410 Mote Sensing Subsystem, Passive Infrared (PIR) Sensor

A set of sensors are under the plastic enclosure of the MSP410 Mote, except the core's board microcontroller and the radio components. These sensors are the system's important interfaces, with an environment responsible for gathering data. The collected information is passed to the board part where a basic manipulation happens. Then the information is encapsulated in a message, which is transmitted through the network forward to the base station.

The PIR sensor supports 360-degree coverage in a horizontal direction. To enhance the sensor's capabilities, a lens is used. The horizontal field of view is further subdivided into nine individual beams. A shadow is produced by warm object close to a sensing element. Whenever a shadow crosses sequentially at least two of the horizontal beams, object detection takes. The four PIR elements also provide "Quad Detect capability." This capability enhances the system's ability to identify an object's movement and direction by including into the data message, which the node sends to the base station, not only the PIR value but also the quad that had the detection.

The outputs of a PIR sensor are affected by the sensor's sensitivity, the sensor's position, the ambient thermal noise and the object's characteristics (type, size, distance, velocity, direction, aspect). Sensors use filtering for the input output signal to increase the sensing performance and to reduce the effect of the noise.

Table 2 summarizes the specification and performance of the MSP410 PIR sensor based on the MSP410 Series User's Manual (Crossbow, 2005).

Specifications - Performance	**Value**	**Comments**
Optical wavelength	5 µm to 14 µm	
Optical bandwidth	0.01 Hz to 15 Hz	
Field of view vertical	± 15° °	
Field of view horizontal	± 45	
Storage temperature	-55°C to +125°C	
Range for human detection	30' to 40'	For Motes height ≈ 3' off the ground Outdoor air temperature ≈ 7°C.
Range for cars detection	50' to 60'	
Range for large tracks detection	70' to 80'	

Table 1. MSP410 Mote PIR Sensor's specification and Performance based on the MSP410 Series User's Manual (Crossbow, 2005).

6. MSP410 Mote Sensing Subsystem, Magnetic Sensor

The magnetic sensor is triggered by changes in the local magnetic field, which may be produced by a near-passing object. The use of proper noise-filtering algorithms and two-stage amplification minimizes false detections and succeeds in maximum detection ranges. The magnetic sensor's specifications are presented in the following table provided by Crossbow in its MSP410 Series User's Manual (Crossbow, 2005).

Parameter	**Typical value**	
Bridge resistance	1100 ohms	
Field range	± 6 gauss (Earth's field = 0.5 gauss)	
Sensitivity	1 mV/V/gauss	
Linearity error (best fit straight line)	± 1 gauss	0.05% FS
	± 3 gauss	0.4% FS
	± 6 gauss	1.6% FS
Bandwidth	DC to 5 MHz	
Noise Density	50 nVsqrt Hz @ 1kHz	
Resolution	120 µgauss @ 50 Hz BW	
Storage Temperature	-55°C to 175°C	

Table 2. MSP410 Mote Magnetic Sensor's specification: MSP410 Series User's Manual (from: Crossbow, 2005)

7. MSP410 Mote Power Characteristics

The MSP410 Mote uses two-AA-battery to be operated. This subsection focuses on some of the power characteristics of the motes. From 3.6 to 2.7 V is the practical operating voltage. Additionally, the MICA2 board can be powered through the 51-pin connector and the two-pin Molex connector. However, in the MSP410CA product, the last three abilities are not applicable because of the enclosure.

Circuit	**Mode**	**Current**
PIR	Off	1 µA
PIR	On	300 µA
Magnetometer, per axis	Off	1 µA
Magnetometer, per axis	On	3 mA
Radio	Off	1 µA
Radio	RX mode	8 mA
Radio at 1 mW	TX mode	16 mA
Processor	Sleep	15 to 20 µA
Processor	Active	8 mA
Serial flash memory	Write	15 mA
Serial flash memory	Read	4 mA
Serial flash memory	Off	2 µA

Table 3. Motes’ power requirements for various operations based on the MSP410 Series and MPR/MIB User’s Manual (Crossbow, 2005)

The above table summarizes the power requirements for various operations. Finally, according to the system’s manual, the two AA batteries last ten hours.

8. MBR410CA Mote Base Station

The MBR410CA Base Station (Figure 8) consists of two different pieces, A MIB510CA serial gateway and a MICA2 series MPR410 radio/processor board.

The base station primarily supports two different operations. First, this configuration allows data aggregation from the nodes on a computer platform connected to the MBR410CA by having the node ID 0 acts as a base station for the WSN,. In addition, the developer has the ability to reprogram the motes by using the RS-232 serial programming interface.

Figure 8. MBR410CA, MSP410CA base station

C. SOFTWARE

1. MOTE-VIEW Client Software

The purpose of mote-view client software is to make the deployment and monitoring of the system easier. It also supports wireless sensor data logging to a database, analysis, and presentation of those data.

Figure 9, The MOTE-VIEW user’s manual image, shows complete three-layer WSN implementation architecture. The client layer provides monitoring interpretation and analysis of the raw data returned by the sensors. The second server layer stores the raw data in a database for logging purposes. Then, those data arrive at MOTE-VIEW through the second-server layer. Finally, in the mote layer, motes use the onboard sensor, and through their program written in TinyOS, perform a specific task, gathering the proper data for the application.

MOTE-VIEW supports all the Crossbow WSN hardware products. MOTE-VIEW was used for checking the MSP410CA systems topological and networking connections during the initial set up monitoring, and evaluation of WSNs.

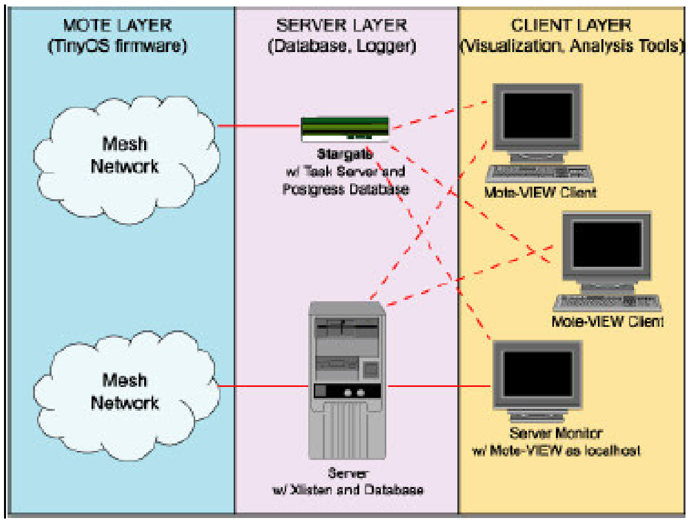
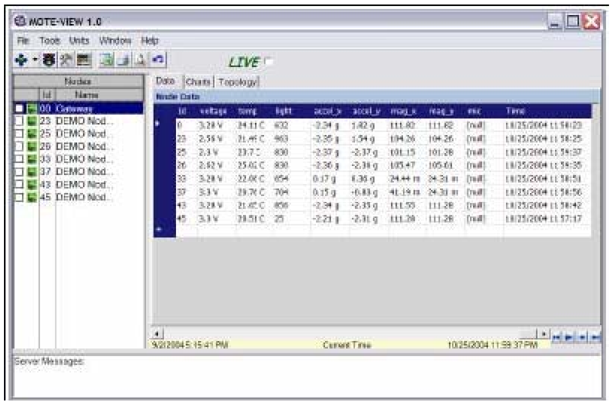

MOTE LAYER
(TinyOS firmware)
SERVER LAYER
(Database, Logger)
CLIENT LAYER
(Visualization, Analysis Tools)
Mesh Network
Stargate w/ Task Server and Posgress Database
Mote-VIEW Client
Mote-VIEW Client
Mesh Network
Server w/ Xlisten and Database
Server Monitor w/ Mote-VIEW as localhost

Figure 9. Three-layer software framework for a WSN: MOTE-VIEW 1.0 User's Manual (from: Crossbow, 2005)

The MOTE-VIEW's main objective is to provide an easy and functional graphical user interface. The following sections provide a brief description of the interface's functionality, as explained in the MOTE-VIEW user's manual (Crossbow, 2005).

Figure 10 shows a screenshot of data returning from the MSP410CA system. Once a sensor network is setup and connected to the user's computer, which runs MOTE-VIEW, the user has the ability, after the proper database and firmware application configuration, or after starting a "data log" menu option, to observe data from the database. Through the window, "data view" tab selection, the user can select the sensor's data, the nodes status, and server's possible messages.



Id	Name
00	Gateway
23	DEMO Nod...
25	DEMO Nod...
26	DEMO Nod...
33	DEMO Nod...
37	DEMO Nod...
43	DEMO Nod...
45	DEMO Nod...

id	voltage	temp	light	accel_x	accel_y	mag_x	mag_y	mic	Time
0	3.28 V	24.11 C	632	-2.34 g	1.82 g	111.82	111.82	(null)	10/25/2004 11:58:23
23	2.56 V	21.49 C	963	-2.35 g	1.54 g	104.26	104.26	(null)	10/25/2004 11:58:25
25	2.3 V	23.7 C	830	-2.37 g	-2.37 g	101.15	101.28	(null)	10/25/2004 11:59:37
26	2.62 V	25.02 C	830	-2.36 g	-2.38 g	105.47	105.61	(null)	10/25/2004 11:59:35
33	3.28 V	22.00 C	654	0.17 g	6.38 g	24.44 m	24.31 m	(null)	10/25/2004 11:58:51
37	3.3 V	29.76 C	704	0.15 g	-0.83 g	41.19 m	24.31 m	(null)	10/25/2004 11:58:56
43	3.28 V	21.45 C	856	-2.34 g	-2.35 g	111.55	111.28	(null)	10/25/2004 11:58:42
45	3.3 V	20.51 C	25	-2.21 g	-2.31 g	111.28	111.28	(null)	10/25/2004 11:57:17

9/2/2004 5:15:41 PM Current Time 10/25/2004 11:59:37 PM

Figure 10. Screenshot presents MOTE-VIEW "Data" view received from MSP410 system: MOTE-VIEW 1.0 User's Manual (from: Crossbow, 2005)

The user also can check different aspects of the sensor network system by selecting different menu options. The "Chart" tab show sensor's historical data graphs: Figure 11 shows the graph of the "Chart" tab. The "Chart" selection can show up to three different graphs from various sensors, and up to twenty-four nodes.

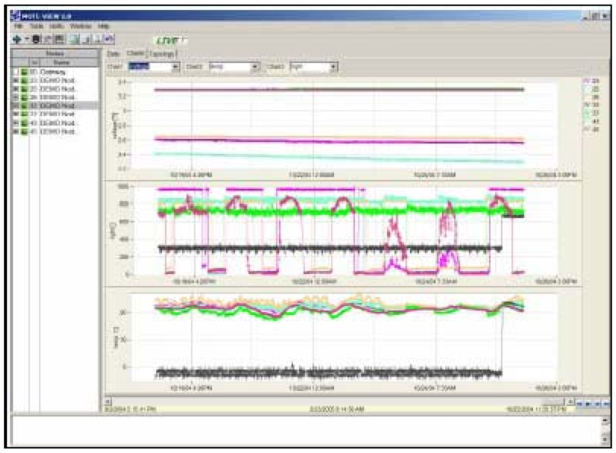


Figure 11. Screenshot presents THE MOTE-VIEW "Chart" view received from the MSP410 system: MOTE-VIEW 1.0 User's Manual (from: Crossbow, 2005)

Figure 12 shows the final MOTE-VIEW presentation option. The "Topology," gives the user the ability to map the network's nodes, including position and parenting information. The user also has the ability to insert background images, presenting properly the system's real development environment. All of the above three charts can be printed by using the print option.

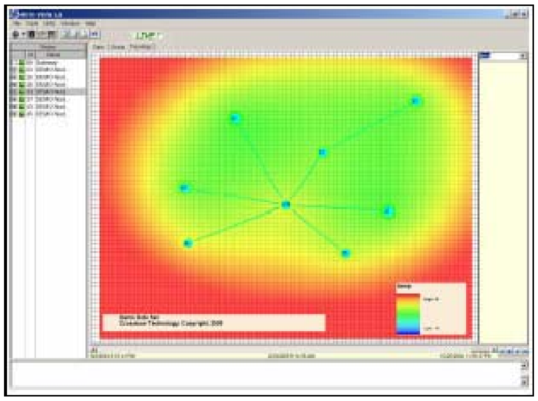


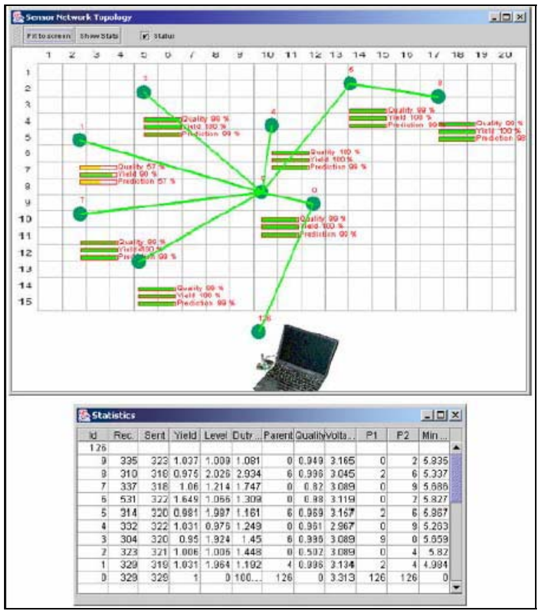
Figure 12. Screenshot presents MOTE-VIEW "Topology" view received from MSP410 system: MOTE-VIEW 1.0 User's Manual (from: Crossbow, 2005)

2. XServe

XServe is a command-line tool that facilitates the sensor’s data readings, which is described in the MOTE-VIEW 1.0 User’s Manual (Crossbow, 2005). The user can use XServe from a Cygwin command line, or as a data-logging server for MOTE-VIEW, using the LogData menu.

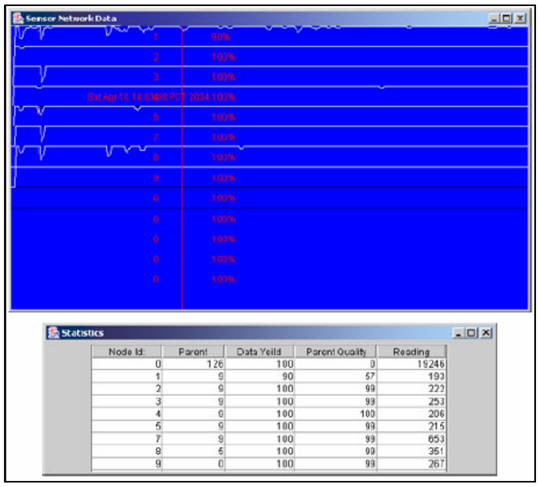
3. Surge Network Viewer (Surge-View)

Surge-View consists of the Surge Graphical User Interface (GUI), the Stats, and the HistoryViewer programs. Although the user can see the sensors’ board data through Surge, main concern of this tool is related to the system’s networking issues. The user can view the mote’s connectivity and routing statistics through the GUI. Additionally, the network performance can be stored in the control station (PC) for later usage. Stats give data about the network’s condition. Finally, HistoryViewer enables the network’s topology and statistics playback. The following figures present different outputs of the Surge-View software product. (Getting started Guide, Crossbow, 2005).



Id	Rec.	Sent	Yield	Level	Dutr...	Parent	Quality	Volta...	P1	P2	Min...
126											
9	335	323	1.037	1.009	1.081	0	0.949	3.165	0	2	5.835
8	310	318	0.975	2.026	2.934	6	0.996	3.045	2	6	5.337
7	337	318	1.06	1.214	1.747	0	0.82	3.089	0	9	5.686
6	531	322	1.649	1.066	1.309	0	0.98	3.119	0	7	5.827
5	314	320	0.981	1.987	1.161	6	0.969	3.167	2	6	5.967
4	332	322	1.031	0.976	1.249	0	0.961	2.967	0	9	5.263
3	304	320	0.95	1.924	1.45	6	0.996	3.089	9	0	5.659
2	323	321	1.006	1.006	1.448	0	0.502	3.089	0	4	5.82
1	329	319	1.031	1.964	1.192	4	0.996	3.134	2	4	4.984
0	329	329	1	0	100...	126	0	3.313	126	126	0

Figure 13. Surge's output for a WSN Topology and Statistics: Getting started Guide (from: Crossbow, 2005).



Node Id:	Parent	Data Yeild	Parent Quality	Reading
0	126	100	0	19246
1	9	90	57	193
2	9	100	99	222
3	9	100	99	253
4	9	100	100	206
5	9	100	99	215
7	9	100	99	653
8	5	100	99	351
9	0	100	99	267

Figure 14. HistoryViewer output for a WSN Data Topology and Statistics (from: Getting started Guide, Crossbow, 2005).

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONFIGURATION AND MANAGEMENT APPLICATION

A. APPLICATION REQUIREMENT

1. Planning Stage

If we deploy sensor nodes without proper planning, it will require a high cost and effort to correct problems later.

First requirement in planning stage is the information about terrain. So, application needs to show user a map or a picture which displays configuration of the ground. Then the user can decide where sensor node should be deployed.

To deploy sensor nodes, application needs to know the mote positions and display the motes on the map by their position.

Next requirement is verifying deployment plan. After deploying each node, user wants to check the deployment. A most important consideration in this step is the connectivity between nodes. Connectivity depends on testing mote, MSP 410CA. Range limit is various by the purposes and situations. Salatas (2005) tested that the average range between motes to be 45M allows them to be connected to one another to form a mesh network. So the user can enter a range for the deployment. Our application needs to know this range and use it to verify the network connectivity.

2. Management Stage

After the planning stage, the user would deploy the network, and should test the network status as deployed. Network status includes connection status, and location and battery level of each node. To check network status, our application needs to receive such information from deployed nodes. This data is received from MBR 410CA mote base station by serial.

B. APPLICATION COMPONENT AND EVALUATION

1. Planning Stage

a. Setting Background

Firstly, user needs to set the background picture which will show the place where sensor node will be deployed. In this application, the map can be a jpeg image of the terrain. To deploy motes, user needs their precise coordinates on the terrain. These coordinates can be collected by a GPS device or other means. In this thesis, we used Google earth [http://earth.google.com/] to collect this information. In our application, the user can choose specific place where mote will be deployed and can check their coordinates. Figure 15 shows picture and coordinates.

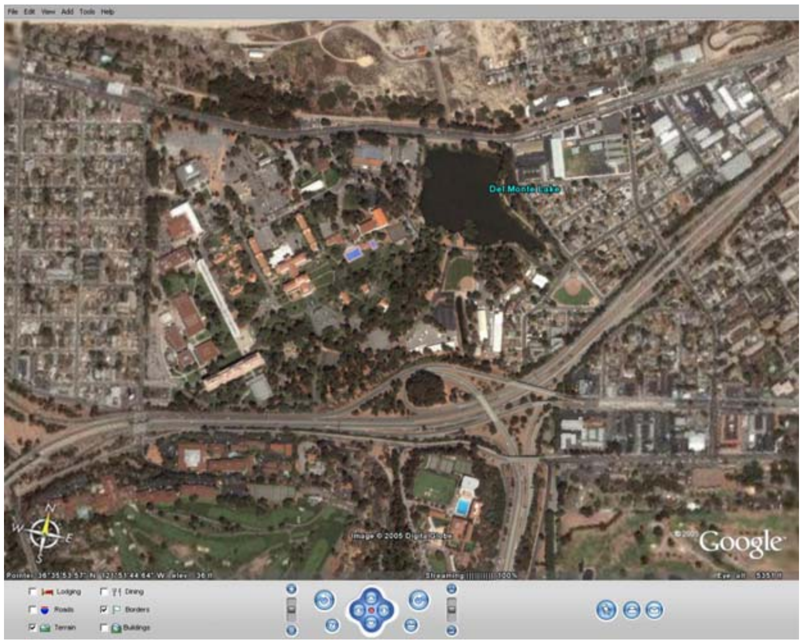
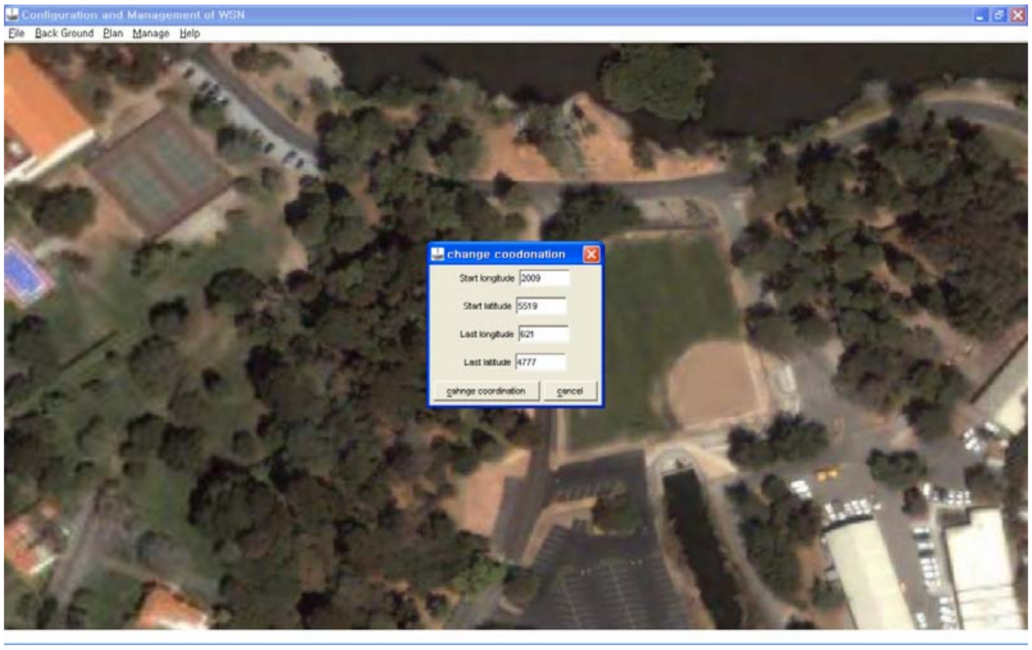


Figure 15. Snapshot of Google earth

User can import the image using the "Background" menu command "import". After importing picture in the configuration and management application, we need to enter coordinates from window to UTM (Universal Transverse Mercater Grid). Figure 16 shows that user input upper-left and lower-right coordinates of the area of deployment. User can open the input box from the "Background" menu command "Set Coordinates". We assume that tested area is small and same degrees and minutes in coordinates. So, user does not need to input direction in latitude and longitude as well as degrees and minutes. For example, upper left corner coordinates of the following picture is actually 121° 52' 20.09" N and 36° 35' 55.19" W. But, the user only needs to input 2009 and 5519. These input coordinates will be saved in file "coordinates.txt".


Configuration and Management of WSN
File Back Ground Plan Manage Help
change coodonation
cahnge coordination
cancel

Start longitude 2009
Start latitude 5519
Last longitude 621
Last latitude 4777

Figure 16. GPS Coordinates Input

After entering the coordinates, user can check motes coordinates by moving cursor over the mote icon on the display.

b. Mote Addition and Deployment

After deciding the place where the user wants to deploy motes, user can input the GPS coordinates of the motes using the "Plan" menu command "Insert mote". These input coordinates will be saved in the file, "data.txt". After the coordinates of motes have been entered, the user can deploy motes by selecting the "Plan" menu command "Deploy Mote". Figure 17 shows deployed mote on the screen.

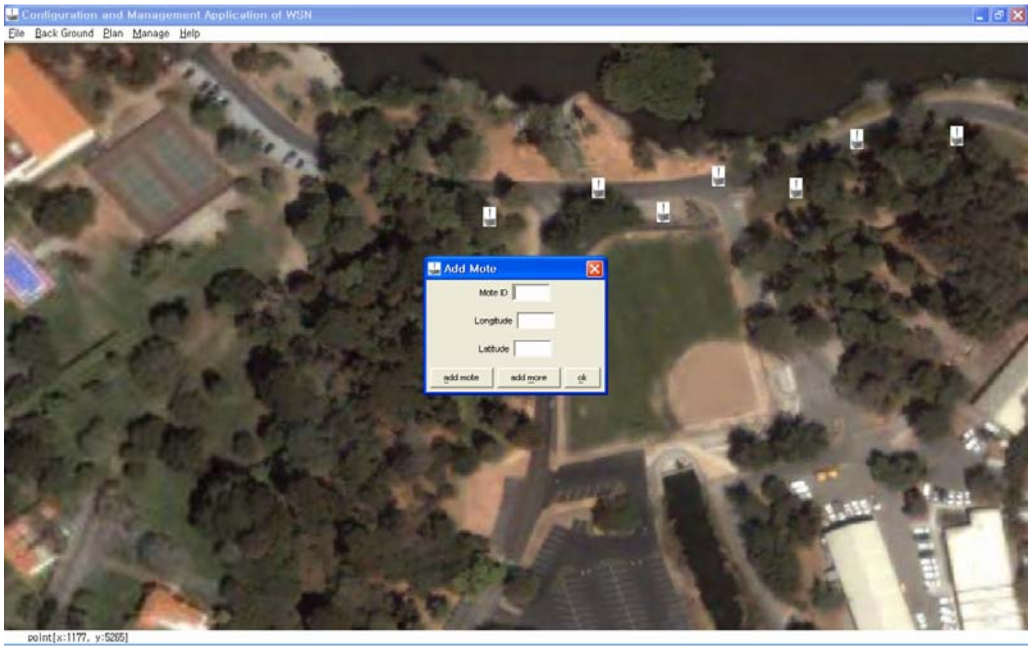

Configuration and Management Application of WSN
File Back Ground Plan Manage Help
Add Mote
Mote ID
Longitude
Latitude
add mote
add more
ok
point[x:1177, y:5265]

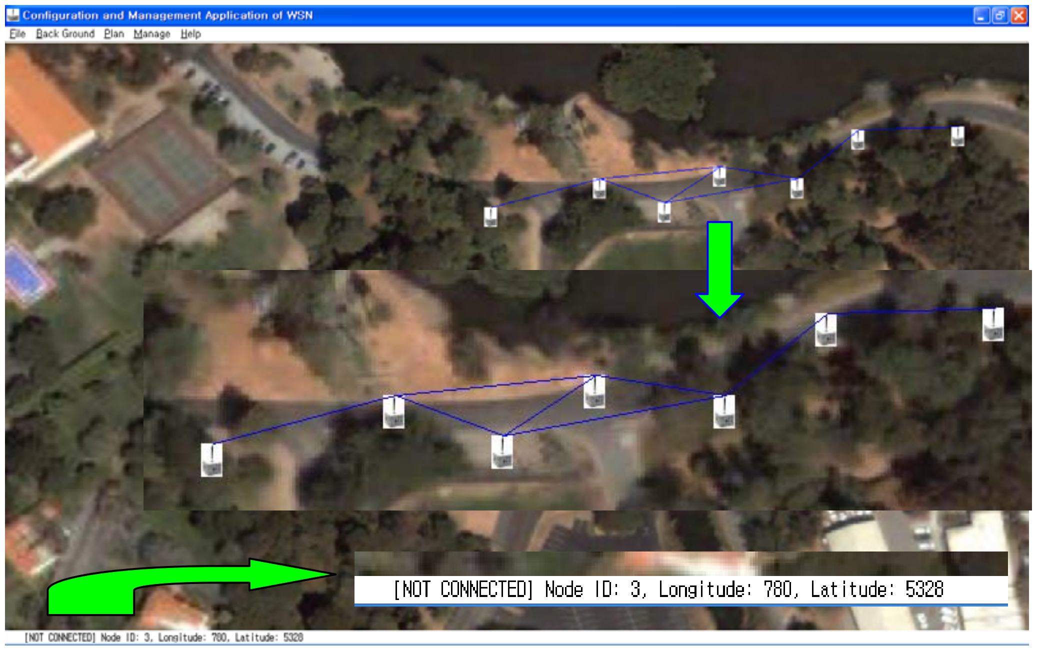
Figure 17. User input for mote deployment and Screen after deploying motes

c. Verification

To verify that the deployed configuration works, user needs to input the range limit of the mote communication. Range limit should vary depending on the geographic layout of the deployment. For example, the range of the mote communication may be smaller in steep slopes..

Whenever the user chooses the "Verify" command from the "Plan" menu, application draws blue line between motes, if the distance between each mote is in the range limit. If there is no line, it means that distance is over the range limit. Figure 18 shows the display upon verification. In this step, calculated distances between two motes are saved to file, "ConfigData.txt" which is used to by Salatas' (2005) Object tracking application.

User can check ID and coordinates of each mote by moving cursor on top of the mote icon. A text-line along the bottom of application window shows this information and connection status. Since our application is not yet connected to MBR 410CA, it shows "NOT CONNECTED".



[NOT CONNECTED] Node ID: 3, Longitude: 780, Latitude: 5328

[NOT CONNECTED] Node ID: 3, Longitude: 780, Latitude: 5328

Figure 18. Verification line between each mote and connection status

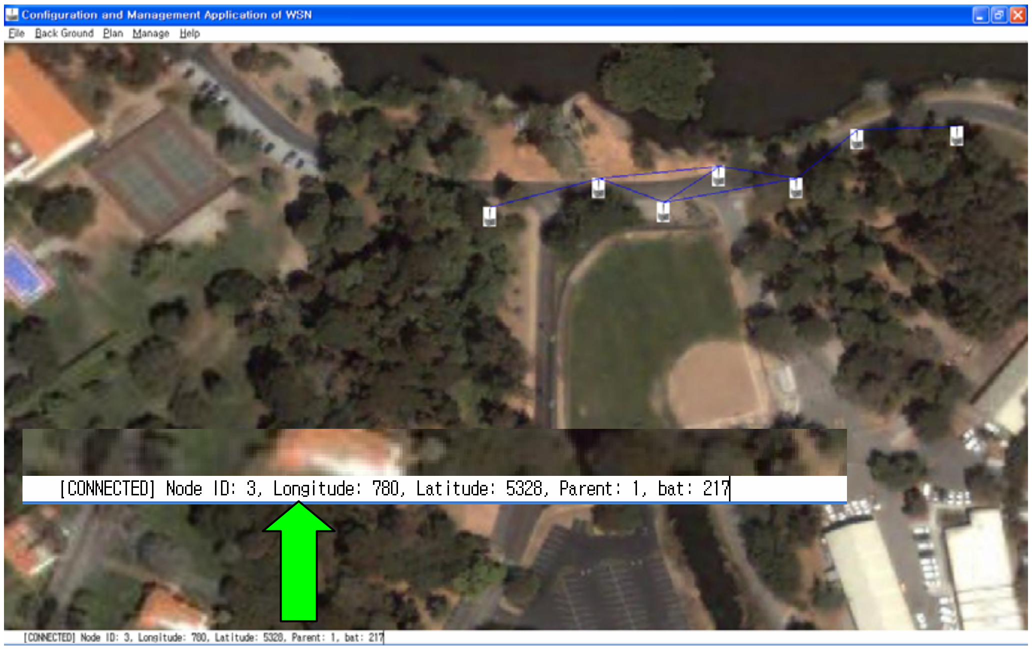
2. Management Stage

a. Receive Data from Base Station

To receive data from MBR 410CA, user can select serial port or converted USB port between laptop and MBR 410CA. By default, COM 5 is set in application.

b. Network Status

Upon connecting to MBR 410CA, the application starts receiving live data from MBR 410CA and saves this data to a file "data.txt". So, when the user clicks each mote, application reads this file and shows received information in the line across the bottom of the screen. Figure 19 shows this information-connection status, node id, longitude, latitude, parent id and battery status.



```
[CONNECTED] Node ID: 3, Longitude: 780, Latitude: 5328, Parent: 1, bat: 217
```

```
[CONNECTED] Node ID: 3, Longitude: 780, Latitude: 5328, Parent: 1, bat: 217
```

Figure 19. Connection status of deployed

This application produces several data files. The data received from MBR410CA base station is important for checking network status of WSN. But this data will grow fast as time goes. So it needs to be archived periodically.

VI. DISCUSSION

A. SUMMARY

In this thesis, we have discussed issues related to the planning, configuration and management of WSNs. For efficient deployment strategies, we have covered general strategies and energy-efficient strategies in Chapter II. General strategies can be divided into predetermined, self-regulated, randomly undermined and biased distribution strategy by purposes and situation.

In Chapter III, we have discussed management challenges. Due to difference between traditional network and WSN, WSN need to consider factors which are not considered in traditional networks. We considered different approaches along the management dimension to deal with complexity of WSN. Management dimension can be divided into three big parts; WSN functionalities, functional areas and management levels.

MSP410CA was described in Chapter IV. MSP410CA consists of eight motes which are in charge of communication and detection, and a base station which is the interface between motes and computer.

We developed a visual application to help configure and deploy WSNs. The developed application has been presented in Chapter V. In our application, the user needs to set background and change coordinates for deploying mote into predetermined geography. Input coordinates for deploying motes in planning stage is saved to file which is used later with GPS to deploy motes in real geography. The data received from MBR410CA base station is also saved to file to monitor network status.

B. CONCLUSIONS AND FUTURE WORK

WSNs are an exciting and useful technology which will be used in various areas in the near future. Though a lot of research has been conducted, there are several open problems to be solved. One of the problems is management of these networks. As mentioned before, traditional networks are quite different with WSNs. We explored challenges and existing methods of configuration and management of WSN. Though we have developed an application for these purposes, we have several additional issues to be investigated. For example, our application assumes that deployed area is not wide area due to range limit in MSP410CA security system. So, user does not need to input all of the coordinates. But, for a wide area, user needs to input all coordinates and application needs to handle these data, i. e., different degrees and minutes. Furthermore for power management, if battery level of mote is low, it needs to give a manager alarm message which will be useful to keep network status. Our future work involves building more advanced applications as well as solving the issues mentioned above and testing them with a real WSN in real life.

APPENDIX: CONFIGURATION AND MANAGEMENT APPLICATION SOURCE CODE.

```
/**

 * <p>Title:              ConfigManageApplication          </p>

 * <p>Description: This class is responsible to provide a simple user
interface.

 *                 </p>

 * <p>Copyright: Copyright (c) 2005</p>

 * <p>Company: Naval Postgraduate School, Monterey, CA</p>

 * @author Min Kim

 * @version 1.0

 */


import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

import java.io.*;


class ConfigManageApplication extends JFrame implements ActionListener,
            MouseListener, MouseMotionListener {

      private JMenu file, bg, plan, manage,help;

      private JMenuItem f1, f2, f3, f4, f5, b1, b2, p1, p2, p3, p4,
                  m1,m2, m3, m4, h;

      private JMenuBar bar;

      private JPanel main;

      private JTextArea tArea;

      private MainScreen screen;

      private AddMote am;
      private ChangeCoordinates co;

      private SimpleRead sr;

      private AboutBox about;

      private PrintJob pjob;


public ConfigManageApplication() {

      super("Configuration and Management Application of WSN");

      init();

}


private void init() {

      try {


      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName()
            );


      setIconImage(Toolkit.getDefaultToolkit().getImage("moteIcon.jpg"))
            ;

            addMouseMotionListener(this);

            addMouseListener(this);

            file = new JMenu("File");

            file.setMnemonic('f');

            f1 = new JMenuItem("New", new ImageIcon("new.jpg"));

            f1.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
                  ActionEvent.CTRL_MASK));

            f2 = new JMenuItem("Open", new ImageIcon("open.jpg"));

            f2.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
                  ActionEvent.CTRL_MASK));

            f3 = new JMenuItem("Save", new ImageIcon("save.jpg"));
```

```
f3.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
      ActionEvent.CTRL_MASK));
f4 = new JMenuItem("Print", new ImageIcon("print.jpg"));
f4.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,
      ActionEvent.CTRL_MASK));
f5 = new JMenuItem("Exit", new ImageIcon("moteIcon.jpg"));
f5.setAccelerator(KeyStroke.getKeyStroke('s',
      ActionEvent.ALT_MASK));

file.add(f1);
file.add(f2);
file.add(f3);
file.addSeparator();
file.add(f4);
file.addSeparator();
file.add(f5);
f1.addActionListener(this);
f2.addActionListener(this);
f3.addActionListener(this);
f4.addActionListener(this);
f5.addActionListener(this);

bg = new JMenu("Back Ground");
bg.setMnemonic('b');
b1 = new JMenuItem("Import", new ImageIcon("moteIcon.jpg"));
b2 = new JMenuItem("Set Coordinates", new
      ImageIcon("moteIcon.jpg"));
bg.add(b1);
bg.add(b2);
```

```
b1.addActionListener(this);

b2.addActionListener(this);


plan = new JMenu("Plan");

plan.setMnemonic('p');

p1 = new JMenuItem("Clear data", new
      ImageIcon("clear.jpg"));

p2 = new JMenuItem("Insert Mote", new
      ImageIcon("addmote.jpg"));

p3 = new JMenuItem("Deploy Mote", new
      ImageIcon("moteIcon.jpg"));

p4 = new JMenuItem("Verify", new ImageIcon("verify.jpg"));

plan.add(p1);

plan.add(p2);

plan.add(p3);

plan.add(p4);

p1.addActionListener(this);

p2.addActionListener(this);

p3.addActionListener(this);

p4.addActionListener(this);


manage = new JMenu("Manage");

manage.setMnemonic('m');

m1 = new JMenuItem("Insert_COM_Number", new
      ImageIcon("moteIcon.jpg"));

m2 = new JMenuItem("Clear data", new
      ImageIcon("moteIcon.jpg"));

m3 = new JMenuItem("Recive data", new
      ImageIcon("moteIcon.jpg"));
```

```
m4 = new JMenuItem("Stop data", new
      ImageIcon("moteIcon.jpg"));

manage.add(m1);

manage.add(m2);

manage.add(m3);

manage.add(m4);

m1.addActionListener(this);

m2.addActionListener(this);

m3.addActionListener(this);

m4.addActionListener(this);


help = new JMenu("Help");

help.setMnemonic('h');

h = new JMenuItem("About..", new ImageIcon("about.jpg"));

help.add(h);

h.addActionListener(this);


bar = new JMenuBar();

bar.add(file);

bar.add(bg);

bar.add(plan);

bar.add(manage);

bar.add(help);


tArea = new JTextArea();

main = new JPanel();

main.setLayout(new BorderLayout());

screen = new MainScreen();
```

```
            main.add(screen, BorderLayout.CENTER);

            main.add(tArea, BorderLayout.SOUTH);

            setContentPane(main);

            setJMenuBar(bar);

            addWindowListener(new WindowAdapter() {

                  public void windowClosing(WindowEvent we) {

                        System.exit(0);

                  }

            });

            about = new AboutBox(this, "Configuration and Management of
                        WSN", true);

            setSize(1009,675);

            setVisible(true);

      } catch (Exception e) {

            e.printStackTrace();

            System.err.print("Sorry..there was an error initializing
                  main window");

      }

}


public void mouseClicked(MouseEvent e) {

      String s, ss;

      int dataId=100;

      int ux=100;

      int uy=100;

      int id=100;

      int par=100;

      int bat=100;
```

```
double x = (e.getX()-4)*co.modifiedX;

double y = (e.getY()-51)*co.modifiedY;

x = co.inputStartLongitude-x;

y = co.inputStartLatitude-y;

Double dx = new Double(x);

Double dy = new Double(y);

int x1 = dx.intValue();

int y1 = dy.intValue();

String str = tArea.getText();

tArea.setText("     point[x:"+x1+", y:"+y1+"]");

try{

      BufferedReader in = new BufferedReader

                   (new FileReader("data.txt"));

      while((s=in.readLine()) != null){

            String[] val = s.split("[:[^0-9]]");

            dataId =  Integer.parseInt(val[3]);

            ux = Integer.parseInt(val[15]);

            uy = Integer.parseInt(val[26]);

            if ((x1<ux)&(x1>ux-25)&(y1<uy)&(y1>uy-25)){

                  tArea.setText("    [NOT CONNECTED] Node ID:
                  "+dataId+", Longitude: "+ux+", Latitude: "+uy);

                  try{

                        BufferedReader inn = new BufferedReader

                        (new FileReader("received data.txt"));

                        while((ss=inn.readLine()) != null){

                              String[] value = ss.split("[:[^0-
                                    9]]");

                              id = Integer.parseInt(value[4]);
```

```
                                par = Integer.parseInt(value[14]);

                                bat = Integer.parseInt(value[21]);

                                if ((x1<ux)&(x1>ux-
                            25)&(y1<uy)&(y1>uy-25)&(id==dataId)){

                                    tArea.setText("    [CONNECTED]
                                    Node ID: "+id+", Longitude:
                                    "+ux+", Latitude: "+uy

                                    +", Parent: "+par+", bat:
                                    "+bat);

                                }

                            }

                        }

                        catch(FileNotFoundException re) {

                            re.printStackTrace();

                        }

                        catch(IOException re) {

                            re.printStackTrace();

                        }

                    }

                }

            }

            catch(FileNotFoundException ef) {

                ef.printStackTrace();

            }

            catch(IOException ef) {

                ef.printStackTrace();

            }

        }
```

```
public void mouseDragged(MouseEvent e) {

}

public void mouseEntered(MouseEvent e) {

}

public void mouseExited(MouseEvent e) {

}

public void mouseMoved(MouseEvent e) {

}

public void mousePressed(MouseEvent e) {

}

public void mouseReleased(MouseEvent e) {

}

public void actionPerformed(ActionEvent actionEvent) {

      if (actionEvent.getSource() == f1) {

            screen.fromStart();

      }

      else if (actionEvent.getSource() == f2) {

            screen.im();

      }

      else if (actionEvent.getSource() == f3) {

            screen.createImage();

            screen.save();

      }

      else if (actionEvent.getSource() == f4) {

            pjob = getToolkit().getPrintJob(this, "Printing", null,
                  null);

            if (pjob != null) {

                  Graphics pg = pjob.getGraphics();
```

```
                        if (pg != null) {
                              screen.printAll(pg);
                              pg.dispose();
                        }
                  pjob.end();
                  }
            }
            else if (actionEvent.getSource() == f5) {
                        System.exit(0);
            }
            else if (actionEvent.getSource() == b1) {
                  screen.setB();
            }
            else if (actionEvent.getSource() == b2) {
                  co = new ChangeCoordinates(this,"change coodonation",true);
            }
            else if (actionEvent.getSource() == p1) {
                  try{
                        PrintWriter newData = new PrintWriter(new
                                                FileWriter("data.txt"));
                        newData.print("");
                        newData.close();
                  }
                  catch(FileNotFoundException k) {
                              k.printStackTrace();
                  }
                  catch(IOException k) {
                              k.printStackTrace();
```

```
      }

}

else if (actionEvent.getSource() == p2) {

      am = new AddMote(this,"Add Mote",true);

}

else if (actionEvent.getSource() == p3) {

      screen.deployMote();

}

else if (actionEvent.getSource() == p4) {

      MainScreen.rangeLimit = JOptionPane.showInputDialog(null,
                              "Insert the range limit");

      screen.verify();

}

else if (actionEvent.getSource() == m1) {

      SimpleRead.comNumber = JOptionPane.showInputDialog(null,

            "insert the con number for the serial port (COM#)");

}

else if (actionEvent.getSource() == m2) {

      try{

            PrintWriter newData = new PrintWriter(new
                              FileWriter("received data.txt"));

            newData.print("");

            newData.close();

      }

      catch(FileNotFoundException k) {

                  k.printStackTrace();

      }

      catch(IOException k) {
```

```
                        k.printStackTrace();
                }
        }
        else if (actionEvent.getSource() == m3) {
                SimpleRead.begin();
                tArea.setText("     Data is receiving from Base Station");
                screen.drawLine();
        }
        else if (actionEvent.getSource() == m4) {
                SimpleRead.pause();
                tArea.setText("     Data receiving is stopped");
        }
        else if (actionEvent.getSource() == h) {
        about.setVisible(true);
        }
}

public static void main(String args[]) {
        ConfigManageApplication fg = new ConfigManageApplication();
}
}
```

```
/**
 * <p>Title:                  MainScreen                  </p>
 * <p>Description: This class is responsible to draw everything.
 *                </p>

 * <p>Copyright: Copyright (c) 2005</p>

 * <p>Company: Naval Postgraduate School, Monterey, CA</p>

 * @author Min Kim

 * @version 1.0

 */


import com.sun.image.codec.jpeg.*;

import java.awt.event.*;

import java.awt.*;

import javax.swing.*;

import java.awt.image.*;

import java.util.Vector;

import java.net.*;

import java.awt.geom.*;

import java.io.*;


class MainScreen extends JPanel {

        private ConfigManageSystem main;

        private ChangeCoordinates co;
        public static final int LINE = 1;

        public static final int PIC = 2;

        private int mode = LINE;

        private int pos;

        private int x1, y1, x2, y2; //calibrate dx, dy to draw line to mote antena

        private int ix1, iy1, ix2, iy2;

        private double ux1, uy1, ux2, uy2;//x, y value get from point rp

        private double dx1, dy1, dx2, dy2;//change double value from dataLongitude,

                dataLatitude

        private double dataLongitude, dataLatitude, dx, dy;

        private Vector all = new Vector();

        private String[] filelist;

        private Image offScreen;

        private Image buffer;

        private JFileChooser fChooser1, fChooser2, fChooser3;

        double modifiedX, modifiedY, modifiedX1, modifiedY1;

        static String rangeLimit = "20";

        Rectangle rrp;


public MainScreen() {
```

```
setBackground(Color.white);

setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));

fChooser1 = new JFileChooser();

fChooser1.setDialogTitle("Insert image..");

fChooser1.setDialogType(JFileChooser.OPEN_DIALOG);

fChooser1.setApproveButtonMnemonic('i');

fChooser1.setFileHidingEnabled(true);

fChooser1.setApproveButtonToolTipText("Click to insert the image");

fChooser2 = new JFileChooser();

fChooser2.setDialogTitle("Save as JPEG");

fChooser2.setDialogType(JFileChooser.SAVE_DIALOG);

fChooser2.setApproveButtonMnemonic('s');

fChooser2.setFileHidingEnabled(true);

fChooser2.setApproveButtonToolTipText("Click to save image as JPEG");

fChooser3 = new JFileChooser();

fChooser3.setDialogTitle("Open image..");

fChooser3.setDialogType(JFileChooser.OPEN_DIALOG);

fChooser3.setApproveButtonMnemonic('i');

fChooser3.setFileHidingEnabled(true);

fChooser3.setApproveButtonToolTipText("Click to open the image");
```

```
        try {

                UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

        } catch (Exception we) {

        }

}


private boolean confirm(String one) {

        one = one.substring(one.lastIndexOf('.'));

        if (one.equalsIgnoreCase(".jpg") || one.equalsIgnoreCase(".jpeg") ||

        one.equalsIgnoreCase(".gif") || one.equalsIgnoreCase(".png"))

                return true;

        else

                return false;

}


public void createBuffer() {

        buffer = createImage(getWidth(), getHeight());

        displayAll(buffer.getGraphics());

}
```

```
public void createImage() {

        offScreen = createImage(getWidth(), getHeight());

        displayAll(offScreen.getGraphics());

}



public void fromStart() {

        all.removeAllElements();

        setBackground(Color.white);

        setForeground(Color.black);

        paint(getGraphics());

}



public void im() {

        if (fChooser3.showDialog(main, "Open image") == JFileChooser.APPROVE_OPTION) {

                if (confirm(fChooser3.getSelectedFile().getName())) {

                        File jmk = fChooser3.getSelectedFile();

                        fromStart();

                        Point lp = new Point(0, 0);

                        try {
```

```
                                        URL jk = new java.net.URL("file:/" +

                                        jmk.getAbsolutePath());

                                        Image pic = Toolkit.getDefaultToolkit().getImage(jk);

                                        MediaTracker med = new MediaTracker(this);

                                        med.addImage(pic, 0);

                                        med.waitForID(0);

                                        int wi = pic.getWidth(this);

                                        int he = pic.getHeight(this);

                                        wi = (getSize().width - wi) / 2;

                                        he = (getSize().height - he) / 2;

                                        lp = new Point(wi, he);

                                        all.addElement(new All(PIC, pic, lp));

                                        paint(getGraphics());

                                } catch (Exception mn) {

                                }

                        }

                }

        }



        public void save() {
```

```
File sav = null;

if (fChooser2.showDialog(main, "Save image") == JFileChooser.APPROVE_OPTION) {

        sav = new File(fChooser2.getCurrentDirectory(),

        fChooser2.getSelectedFile().getName() + ".jpg");

        BufferedImage bimg = null;

        int w = offScreen.getWidth(this);

        int h = offScreen.getHeight(this);

        int[] pixels = new int[w * h];

        PixelGrabber pg = new PixelGrabber(offScreen, 0, 0, w, h, pixels, 0, w);

        try {

                pg.grabPixels();

        } catch (InterruptedException ie) {

                System.err.println("Sorry..error parsing Image");

        }

        bimg = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

        bimg.setRGB(0, 0, w, h, pixels, 0, w);

        try {

                FileOutputStream fos = new FileOutputStream(sav);

                JPEGImageEncoder jpeg = JPEGCodec.createJPEGEncoder(fos);

                jpeg.encode(bimg);
```

```
                    fos.close();

                } catch (Exception jk) {

                    System.err.println("Sorry..error writing to file");

                }

        }

}



//method which draw mote by user input

public void deployMote(){

        String string,str, stringModifiedX, stringModifiedY;

        Image image;

        int dataId, drawLongitude, drawLatitude;

//get image from file

        image = Toolkit.getDefaultToolkit().getImage("mote.JPG");

        MediaTracker dTracker = new MediaTracker(this);

        Point np = new Point(0, 0);

        try{

                dTracker.addImage(image,0);

                dTracker.waitForAll();

        }
        catch(InterruptedException mn){

        }

//reading coordinates from input file

        try{

                BufferedReader in = new BufferedReader(new FileReader("data.txt"));

                while((string=in.readLine()) != null){

                        String[] value = string.split("[:[^0-9]]");

                        dataId =  Integer.parseInt(value[3]);

                        dataLongitude = Integer.parseInt(value[15]);

                        dataLatitude = Integer.parseInt(value[26]);

                        System.out.println("dataId:"+dataId);

                        System.out.println("dataLongitude:"+dataLongitude);

                        System.out.println("dataLatitude:"+dataLatitude);



                        try{

                                BufferedReader reader = new BufferedReader

                                        (new FileReader("coordinates.txt"));

                                while((str=reader.readLine()) != null){

                                        String[] coValue = str.split("[:[^0-9]]");

                                        modifiedX1 = Double.parseDouble(coValue[0]);
```

```
                    modifiedY1 = Double.parseDouble(coValue[2]);

                    stringModifiedX = coValue[4]+"."+coValue[5];

                    stringModifiedY = coValue[6]+"."+coValue[7];

                    modifiedX = Double.parseDouble(stringModifiedX);

                    modifiedY = Double.parseDouble(stringModifiedY);

                    dataLongitude = (modifiedX1-

                              dataLongitude)/modifiedX;

                    dataLatitude = (modifiedY1-

                              dataLatitude)/modifiedY;

                    System.out.println("modifiedX1:"+modifiedX1);

                    System.out.println("smodifiedX:"+stringModifiedX);

                    System.out.println("dataLongitude:"+

                              dataLongitude);

                    Double dx= new Double(dataLongitude);

                    Double dy= new Double(dataLatitude);

                    drawLongitude = dx.intValue();

                    drawLatitude = dy.intValue();


//draw mote by input coordinates

                    np = new Point(drawLongitude, drawLatitude);
                    all.addElement(new All(PIC, image, np));
                    paint(getGraphics());
                }
            }
            catch(FileNotFoundException e) {
                e.printStackTrace();
            }
            catch(IOException e) {
                e.printStackTrace();
            }
        }
    }
    catch(FileNotFoundException e) {
        e.printStackTrace();
    }
    catch(IOException e) {
        e.printStackTrace();
    }
}
```

```
public void verify(){

        String r; //string which is reading data file

        double dx, dy; //parsing string which is in string r

        int longitude, latitude; //parsing string which is in string r

        int i = 0;

        double x, y;  //distance between x, y

        double distance = 0;  //distance between point

        int intDistance = 0;

        int preDistance = 0;

        double limit = Double.parseDouble(rangeLimit);

        limit = limit * 3.0824;

        Point [] ip = new Point[9];

        Point2D.Double [] dp = new Point2D.Double[9];



//receive coordinates from data file

        try{

                BufferedReader in = new BufferedReader(new FileReader("data.txt"));

                while((r=in.readLine()) != null){

                        String[] value = r.split("[:[^0-9]]");

                        dx = Double.parseDouble(value[15]);
        dy = Double.parseDouble(value[26]);

        longitude = Integer.parseInt(value[15]);

        latitude = Integer.parseInt(value[26]);

        ip[i] = new Point(longitude, latitude);

        dp[i] = new Point2D.Double(dx, dy);

        System.out.println("ip[" +i+"]:" +ip[i]);

        i++;

}

try{

        PrintWriter newData = new PrintWriter(new

                        FileWriter("/j2sdk1.4.1_02/bin/

                        configuration_file/ConfigData.txt"));

        newData.print("");

        newData.close();

}

catch(FileNotFoundException k) {

                k.printStackTrace();

}

catch(IOException k) {

                k.printStackTrace();
```

```
                    }

//calcurate distance by input coordiantion

                    for (int m = 0; m<i ; m++ ){

                            for (int n = m+1 ; n<i ; n++ ){

                                    x = Math.pow(Math.abs(dp[m].getX()-dp[n].getX()),2);

                                    y = Math.pow(Math.abs(dp[m].getY()-dp[n].getY()),2);

                                    distance=           Math.sqrt(x+y);

                                    Double doubleDistance= new Double(distance);

                                    intDistance = doubleDistance.intValue();

                                    String temp="";

                                    String ttemp="";

                                    String configData="";

                                    if ((m+1)==n)     {


                                            /*ConfigData.txt saves  user's coordinates input.

                                              This file is used in object tracking application.

                                            */

                                            try{

                                                    BufferedReader configReader = new

                                                            BufferedReader  (new FileReader
        ("/j2sdk1.4.1_02/bin/

        configuration_fil/ConfigData.txt"));

//existing input case

while((temp=configReader.readLine()) !=
        null){

        configData = m+","+preDistance+

                        ","+intDistance;

        ttemp=ttemp+"\n"+temp;

}

preDistance = intDistance;

configData = configData+ttemp;;

System.out.println("all
input"+"\n"+configData);

try{

        FileWriter dataOutput = new
                FileWriter("/j2sdk1.4.1_02/

                bin/configuration_file

                /ConfigData.txt");

        dataOutput.write(configData);
        System.out.println("input to
        file"+"\n"+configData);
        dataOutput.close();
} catch(FileNotFoundException k) {
        k.printStackTrace();
} catch(IOException k) {
        k.printStackTrace();
}
//no input case
while((temp=configReader.readLine()) ==
                                        null){
        temp = m+",0,"+intDistance;
        preDistance = intDistance;
        System.out.println(temp);
        try{
                FileWriter dataOutput = new
                  FileWriter("/j2sdk1.4.1_
                  02/bin/configuration_
                   file/ConfigData.txt");
                dataOutput.write(temp);
```

```
                                        dataOutput.close();
                                } catch(FileNotFoundException k) {
                                        k.printStackTrace();
                                } catch(IOException k) {
                                        k.printStackTrace();
                                }
                        }
                }
                catch(FileNotFoundException ev) {
                        ev.printStackTrace();
                } catch(IOException ev) {
                        ev.printStackTrace();
                }
        }
        //draw blue lines if range limit is not over distance
        between each mote
        ux1 = ip[m].getX();
        uy1 = ip[m].getY();
        ux2 = ip[n].getX();
        uy2 = ip[n].getY();
```

```
}

                    ux1 = (modifiedX1-ux1)/modifiedX ;

                    uy1 = (modifiedY1-uy1)/modifiedY ;

                    ux2 = (modifiedX1-ux2)/modifiedX ;

                    uy2 = (modifiedY1-uy2)/modifiedY;

                    Double dx1= new Double(ux1);

                    Double dy1= new Double(uy1);

                    Double dx2= new Double(ux2);

                    Double dy2= new Double(uy2);

                    x1 = dx1.intValue()+9;

                    y1 = dy1.intValue();

                    x2 = dx2.intValue()+9;

                    y2 = dy2.intValue();

                    rrp = new Rectangle(x1, y1, x2, y2);

                    if (distance<=limit){

                    rrp = new Rectangle(x1, y1, x2, y2);

                    all.addElement(new All(LINE, Color.blue, rrp));

                    paint(getGraphics());

                    }

          }

}
```

```
        }
        catch(FileNotFoundException e) {
                e.printStackTrace();      }
        catch(IOException e) {
                e.printStackTrace();
        }
}

public void drawLine(){


}


public void insertImage(File name, Point loc, String file) {
        try {
                URL jk = new java.net.URL("file:/" + name.getAbsolutePath());
                Image pic = Toolkit.getDefaultToolkit().getImage(jk);
                MediaTracker med = new MediaTracker(this);
                med.addImage(pic, 0);
                med.waitForID(0);
                all.addElement(new All(PIC, pic, loc));
                paint(getGraphics());

        } catch (Exception mn) {

        }

}



//method for importing background

public void setB() {

        try {

                File ju = null;

                String name = "";

                if (fChooser1.showDialog(main, "Insert background") ==

                                                JFileChooser.APPROVE_OPTION) {

                        name = fChooser1.getSelectedFile().getName();

                        if (confirm(name)) {

                                ju = new File(fChooser1.getCurrentDirectory(),

                                fChooser1.getSelectedFile().getName());

                        }

                }

                URL jk = new java.net.URL("file:/" + ju.getAbsolutePath());

                Image pic = Toolkit.getDefaultToolkit().getImage(jk);
                MediaTracker med = new MediaTracker(this);

                med.addImage(pic, 0);

                med.waitForID(0);

                int w = getSize().width;

                int h = getSize().height;

                Image resizedImage = null;

                ImageFilter replicate = new ReplicateScaleFilter(w, h);

                ImageProducer prod = new FilteredImageSource(pic.getSource(),replicate);

                resizedImage = createImage(prod);

                all.insertElementAt(new All(PIC, resizedImage, new Point(0, 0)), 0);

                paint(getGraphics());

        } catch (Exception mj) {

        }

        System.out.println("Done");

}



public void paint(Graphics g) {

        createBuffer();

        g.drawImage(buffer, 0, 0, this);

}
```

```
public void setDrawMode(int mode) {

        this.mode = mode;

        setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));

}



public void update(Graphics g) {

        paint(g);

}



public void displayAll(Graphics g) {

        g.setColor(getBackground());

        g.fillRect(0, 0, getWidth(), getHeight());

        for (int i = 0; i < all.size(); i++) {

                All temp = (All) all.elementAt(i);

                Rectangle p = temp.r;

                switch (temp.type) {

                        case LINE :

                                g.setColor(temp.clr);

                                if (p.width != -1) {
```

```
                        g.drawLine(p.x, p.y, p.width, p.height);
                    } else {
                        g.drawLine(p.x, p.y, p.x, p.y);
                    }
                    break;
                case PIC :
                    g.drawImage(temp.img, temp.loc.x, temp.loc.y, this);
                    break;
            }
        }
        g.dispose();
    }
}


/**
 * <p>Title:        AddMote       </p>
 * <p>Description: This class is responsible to read user's coordination inputs.
 *                 The inputs are saved to file, [data.txt].
 *                 The file is used to draw mote to screen depending on user's inputs.
 *              </p>

 * <p>Copyright: Copyright (c) 2005</p>

 * <p>Company: Naval Postgraduate School, Monterey, CA</p>

 * @author Min Kim

 * @version 1.0

 */


import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.util.Vector;

import java.io.*;


class AddMote extends JDialog implements ActionListener {

        private JPanel idPan, altPan, latPan, p;

        private JButton addB, moreB, okB;

        private JTextField idT, altT, latT;

        private JLabel idL, altL, latL;

        private String inData;
```

```
public AddMote(Frame f, String title, boolean modal) {

        super(f, title, modal);



        idPan = new JPanel();

        altPan = new JPanel();

        latPan = new JPanel();



        addB = new JButton("add mote");

        addB.setMnemonic('a');

        addB.addActionListener(this);

        moreB = new JButton("add more");

        moreB.setMnemonic('m');

        moreB.addActionListener(this);

        okB = new JButton("ok");

        okB.setMnemonic('o');

        okB.addActionListener(this);



        idT = new JTextField(6);

        altT = new JTextField(6);

        latT = new JTextField(6);
```

```
idL = new JLabel("Mote ID");

altL = new JLabel("Longitude");

latL = new JLabel("Latitude");



p = new JPanel(new FlowLayout());

p.add(addB);

p.add(moreB);

p.add(okB);



getContentPane().setLayout(new GridLayout(4, 1));

idPan.add(idL);

idPan.add(idT);

altPan.add(altL);

altPan.add(altT);

latPan.add(latL);

latPan.add(latT);

getContentPane().add(idPan);

getContentPane().add(altPan);

getContentPane().add(latPan);
```

```
          getContentPane().add(p);

          addWindowListener(new WindowAdapter() {

                    public void windowClosing(WindowEvent we) {

                              dispose();

                    }

          });

          pack();

          Dimension scr = Toolkit.getDefaultToolkit().getScreenSize();

          setLocation((scr.width - getSize().width) / 2, (scr.height - getSize().height)

                                        / 2);

          setVisible(true);

}



public void actionPerformed(ActionEvent e) {



          String temp="";

          String ttemp="";

          if (e.getSource() == addB) {

          //method which add first input to next input

                    try{
```

```
                    BufferedReader in = new BufferedReader

                                        (new FileReader("data.txt"));

                    inData = "id:"+idT.getText()+", "+"longitude:" +altT.getText()+",

                "+ "latitude:"+latT.getText();

                    System.out.println("input from dialogue:"+inData);

//existing input case

                    while((temp=in.readLine()) != null){

                            System.out.println("first input:"+temp);

                            ttemp=ttemp+"\n"+temp;

                    }

                    System.out.println("next input:"+ttemp);

                    inData = inData+ttemp;;

                    System.out.println("all input"+"\n"+inData);

                    try{

                        FileWriter dataOutput = new FileWriter("data.txt");

                        dataOutput.write(inData);

                        System.out.println("input to file"+"\n"+inData);

                        dataOutput.close();

                    } catch(FileNotFoundException k) {

                        k.printStackTrace();
                    } catch(IOException k) {

                              k.printStackTrace();

                    }

//no input case

                    while((temp=in.readLine()) == null){

                                        temp = inData;

                                        try{

                                                  FileWriter dataOutput = new

                                                                      FileWriter("data.txt");

                                                  dataOutput.write(temp);

                                                  dataOutput.close();

                                        } catch(FileNotFoundException k) {

                                                  k.printStackTrace();

                                        } catch(IOException k) {

                                                  k.printStackTrace();

                                        }

                    }

          }

          catch(FileNotFoundException ev) {

                    ev.printStackTrace();
                    } catch(IOException ev) {

                            ev.printStackTrace();

                    }

            } else if (e.getSource() == moreB) {

                    idT.setText( "");

                    altT.setText("");

                    latT.setText ("");

            } else if (e.getSource() == okB) {

                                    dispose();

            } else

                    return;

    }

    }



    /**

     * <p>Title:        ChangeCoordination      </p>

     * <p>Description: This class is responsible to read user input coordination.

     *                 and change it to UTM coordination.

                       Then it saves this coordonation to file coordination.txt
```

```
 *              </p>

 * <p>Copyright: Copyright (c) 2005</p>

 * <p>Company: Naval Postgraduate School, Monterey, CA</p>

 * @author Min Kim

 * @version 1.0

 */


import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.util.Vector;

import java.io.*;


class ChangeCoordinates extends JDialog implements ActionListener {

        private JPanel x1Pan, x2Pan, y1Pan, y2Pan, P;

        private JButton changeB, cancelB;

        private JTextField x1T, x2T, y1T, y2T;

        private JLabel x1L, x2L, y1L, y2L;

        protected double inputStartLongitude, inputEndLongitude, inputStartLatitude,
                        inputEndLatitude;
        protected double modifiedX,modifiedY;



public ChangeCoordinates(Frame f, String title, boolean modal) {

        super(f, title, modal);



        x1Pan = new JPanel();

        x2Pan = new JPanel();

        y1Pan = new JPanel();

        y2Pan = new JPanel();



        changeB = new JButton("cahnge coordinates");

        changeB.setMnemonic('c');

        changeB.addActionListener(this);



        cancelB = new JButton("cancel");

        cancelB.setMnemonic('c');

        cancelB.addActionListener(this);



        x1T = new JTextField(8);

        x2T = new JTextField(8);
```

```
y1T = new JTextField(8);

y2T = new JTextField(8);


x1L = new JLabel("Start longitude");

x2L = new JLabel("Last longitude");

y1L = new JLabel("Start latitude");

y2L = new JLabel("Last latitude");


P = new JPanel(new FlowLayout());

P.add(changeB);

P.add(cancelB);


getContentPane().setLayout(new GridLayout(5, 1));

x1Pan.add(x1L);

x1Pan.add(x1T);

x2Pan.add(x2L);

x2Pan.add(x2T);

y1Pan.add(y1L);

y1Pan.add(y1T);

y2Pan.add(y2L);
```

```
		y2Pan.add(y2T);

		getContentPane().add(x1Pan);

		getContentPane().add(y1Pan);

		getContentPane().add(x2Pan);

		getContentPane().add(y2Pan);

		getContentPane().add(P);

		addWindowListener(new WindowAdapter() {

			public void windowClosing(WindowEvent we) {

				dispose();

			}

		});

		pack();

		Dimension scr = Toolkit.getDefaultToolkit().getScreenSize();

		setLocation((scr.width - getSize().width) / 2, (scr.height - getSize().height)
				/ 2);

		setVisible(true);

	}


	public void actionPerformed(ActionEvent e) {
```

```
if (e.getSource() == changeB) {

        inputStartLongitude = Double.parseDouble(x1T.getText());

        inputEndLongitude = Double.parseDouble(x2T.getText());

        inputStartLatitude = Double.parseDouble(y1T.getText());

        inputEndLatitude = Double.parseDouble(y2T.getText());

        modifiedX = (inputStartLongitude-inputEndLongitude)/1000;

        modifiedY = (inputStartLatitude-inputEndLatitude)/600;

        String s = inputStartLongitude + "." + inputStartLatitude + "." +

        modifiedX + "." + modifiedY;

                try{

                        FileWriter dataOutput = new

                                                FileWriter("coordinates.txt");

                                dataOutput.write(s);

                                dataOutput.close();

                } catch(FileNotFoundException k) {

                        k.printStackTrace();

                } catch(IOException k) {

                        k.printStackTrace();

                }

        System.out.println("inputStartLongitude="+inputStartLongitude);
```

```
                System.out.println("inputStartLatitude="+inputStartLatitude);

                System.out.println("inputEndLongitude="+inputEndLongitude);

                System.out.println("inputEndLatitude="+inputEndLatitude);

                System.out.println("modifiedX="+modifiedX);

                System.out.println("modifiedY="+modifiedY);

                dispose();

        } else if (e.getSource() == cancelB) {

                dispose();

        } else

                return;


}

}




/**

 *

 * <p>Title: SimpleRead </p>

 * <p>Description: This class is used as a Serial Port reader for the Object
 *               Tracking application. It opens a serial port connection

 *               and a related input stream, and it read the raw data (bytes)

 *               returned from the Crossbow WSN system

 *               MSP410. Then it extract the useful information based on the

 *               message format that the raw data have and place them

 *               inside an array in order to be available from the rest

 *               software components of the application. The part of this class

 *               that reads data from the serial port is based

 *               on the SimpleRead.java file provided from the following URL

 *  * ref: java.sun.com/developer/ releases/javacomm/SimpleRead.java May 2005

 *               </p>

 * <p>Copyright: Copyright (c) 2005</p>

 * <p>Company: Naval Postgraduate School, Monterey, CA</p>

 * @author Vlasios Salatas

 * @modified by Min Kim

 * @version 1.0

 */




import java.io.*;
import java.util.*;

import javax.comm.*;

import javax.swing.Timer;



public class SimpleRead implements Runnable, SerialPortEventListener {

  static CommPortIdentifier portId;

  static Enumeration portList;



  InputStream inputStream;

  SerialPort serialPort;

  static Thread readThread;

  int numBytes;

  static String comNumber = "COM5";



  int counter;

  // buffer to store the incomming messages

  byte[] holdArray = new byte[1000];

  //array used to store and send the proper data

  int[] dataArray = new int[8];
```

```
//flag to control the program

static boolean flag = true;



/**

 * <p>Title: begin </p>

 * <p>Description: it open the com port and start reading data

 */

public static void begin() {

  portList = CommPortIdentifier.getPortIdentifiers();

  flag = true;

  while (portList.hasMoreElements()) {

    portId = (CommPortIdentifier) portList.nextElement();

    if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {

      if (portId.getName().equals(comNumber)) {

        SimpleRead reader = new SimpleRead();

      }

    }

  }

}
```

```
public static void pause(){

  flag = false;

}




public static void resume(){

  flag = true;

}




public static void exit(){

  //exit the system

  System.exit(0);

}



/**

 * Constructor declaration

 */

public SimpleRead() {
```

```
try {

  serialPort = (SerialPort) portId.open("SimpleReadApp", 20000);

} catch (PortInUseException e) {System.out.println("port in use!!!!!!");}

try {

  inputStream = serialPort.getInputStream();

} catch (IOException e) {}

try {

  serialPort.addEventListener(this);

} catch (TooManyListenersException e) {}

serialPort.notifyOnDataAvailable(true);

try {

  serialPort.setSerialPortParams(57600,

                                 SerialPort.DATABITS_8,

                                 SerialPort.STOPBITS_1,

                                 SerialPort.PARITY_NONE);



  serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);



} catch (UnsupportedCommOperationException e) {}
```

```
  readThread = new Thread(this);

  readThread.start();

}




public void run() {

  while ( flag == true) {

    try {

      Thread.sleep(2000);

    }

    catch (InterruptedException e) {}

  }

}


/**

 * Title: serialEvent

 * Description: it sets the serial port parameters and places the data

 *              in a buffer

 * @param event

 */
```

```
public void serialEvent(SerialPortEvent event) {
  switch(event.getEventType()) {
    case SerialPortEvent.BI:


    case SerialPortEvent.OE:
    case SerialPortEvent.FE:
    case SerialPortEvent.PE:
    case SerialPortEvent.CD:


    case SerialPortEvent.CTS:
    case SerialPortEvent.DSR:
    case SerialPortEvent.RI:
    case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
      break;
    case SerialPortEvent.DATA_AVAILABLE:
      try {
        while (inputStream.available() > 0 && flag == true) {
          byte[] readBuffer = new byte[41];
          numBytes = inputStream.read(readBuffer);
          // Passes the data that the buffer holds into the message method
          message(readBuffer);
        }
      } catch (IOException e) {}
      break;
  }
}


/**
 * <p>Title: message </p>
 * <p>Description: This method first reconstract the message that the MSP410
 *                nodes send through the gateway to the serial port. When the
 *                message is completed it reads and places the importand
 *                values to an array and passes them to the rest software
 *                components of the Tracking Object application
 *
 * @param readBuffer byte[]
 */
private void message(byte[] readBuffer){
  // data variables
  int SeqNumber;
```

```
    int SeqNumberF;

    int vref;

    int quad;

    int pir;

    int mag;

    int audio;

            String s;

            String temp="";

            String ttemp="";

    // it initializes a motionDetector object that it is used later to pass the

    // the data to the motionDetector class.

//    motionDetector detector = new motionDetector();



    // reconstracts the message

    if ( readBuffer[0] == 126 ){

      counter = 0;

      for( int i = 0; i < numBytes; i++){

        holdArray[counter] = readBuffer[i];

        if (i >= 28 && readBuffer[i] == 126) {

          counter = 0;
    }

    counter++;

  }

}

// it stores the data into the data array

else{

  for( int k = 0; k < numBytes; k++){

    holdArray[counter] = readBuffer[k];

    if (counter == 38 && readBuffer[k] == 126) {

      //store the nodeid

      dataArray[0] = unsigned_int(holdArray[11]);

      //store the parentid

      dataArray[1] = unsigned_int(holdArray[19]);



      //calcutate the seq# and store it

      SeqNumber = unsigned_int(holdArray[13]) +

          256 * unsigned_int(holdArray[14]);

      SeqNumberF = unsigned_int(holdArray[20]) +

          256 * unsigned_int(holdArray[21]);

      dataArray[2] = SeqNumberF;
```

```
//calcutate the vref and store it

vref = unsigned_int(holdArray[22]);

dataArray[3] = vref;



//calcutate the quad1 and store it

quad = unsigned_int(holdArray[23]);

dataArray[4] = quad;



//calcutate the pir and store it

pir = unsigned_int(holdArray[24]) +

   256 * unsigned_int(holdArray[25] & 0x03);

dataArray[5] = pir;



//calcutate the mag and store it

mag = unsigned_int(holdArray[26]) +

   256 * unsigned_int(holdArray[27] & 0x03);

dataArray[6] = mag;



//calcutate the audio and store it
```

```
        audio = unsigned_int(holdArray[28]) +
            256 * unsigned_int(holdArray[29] & 0x03);
        dataArray[7] = audio;
        counter = 0;


/*Receiving data from base station is saved to file, [received data.txt].
* It needs to keep network hystory.*/
                try{
                        BufferedReader in = new BufferedReader
                                        (new FileReader("received data.txt"));
                        s = "id: " + dataArray[0] + "  parent: " + dataArray[1] +
                  " vref: " + dataArray[3] ;
                        System.out.println("receved data:"+s);
                        while((temp=in.readLine()) != null){
                                ttemp=ttemp+"\n"+temp;
                        }
                        s = s+ttemp;;
                        try{
                                FileWriter dataOutput = new FileWriter("received
                                                        data.txt");
}

        dataOutput.write(s);

        dataOutput.close();

} catch(FileNotFoundException ke) {

        ke.printStackTrace();

} catch(IOException ke) {

        ke.printStackTrace();

}

while((temp=in.readLine()) == null){

                temp = s;

                try{

                        FileWriter dataOutput = new
                                FileWriter("received data.txt");

                        dataOutput.write(temp);

                        dataOutput.close();

                } catch(FileNotFoundException ke) {

                        ke.printStackTrace();

                } catch(IOException ke) {

                        ke.printStackTrace();

                }

}
```

```
                    }

                    catch(FileNotFoundException ev) {

                         ev.printStackTrace();

                    } catch(IOException ev) {

                         ev.printStackTrace();

                    }


        }//end if

        counter++;

      }//end for k

    }//end else

}//end method


/**

 * Description: It convert the received integer value to unsigned int

 * @param nb

 * @return

 */

 static int unsigned_int(int nb){

   if(nb >= 0 )
```

```
      return nb;

    else

      return(256+nb);

  }



}




/**

 * <p>Title:           AboutBox                    </p>

 * <p>Description: This class is responsible to provide application information.

 *                </p>

 * <p>Copyright: Copyright (c) 2005</p>

 * <p>Company: Naval Postgraduate School, Monterey, CA</p>

 * @author Min Kim

 * @version 1.0

 */



import java.awt.event.*;

import javax.swing.*;
import java.awt.*;


class AboutBox extends JDialog implements ActionListener {

        private JLabel app, ver, cop, ico;

        private JPanel p1, p2, p3;

        private JButton ok;

public AboutBox(Frame par, String title, boolean mode) {

        super(par, title, mode);

        getContentPane().setLayout(new BorderLayout());

        p1 = new JPanel(new BorderLayout());

        p2 = new JPanel(new FlowLayout());

        ok = new JButton("Ok");

        ok.addActionListener(this);

        ok.setMnemonic('o');

        p2.add(ok);

        p3 = new JPanel(new GridLayout(4, 1));

        app = new JLabel("Application for Configuration and Management of WSN");

        ver = new JLabel("Version 1.0");

        cop = new JLabel("Copyright 2005 by Min Kim");

        p3.add(app);
```

```
p3.add(ver);

p3.add(cop);

p1.add(p2, BorderLayout.SOUTH);

p1.add(p3, BorderLayout.CENTER);

getContentPane().add(p1, BorderLayout.CENTER);

try {

        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

} catch (Exception fg) {

}

pack();

addWindowListener(new WindowAdapter() {

        public void windowClosing(WindowEvent we) {

                dispose();

        }

});

app.setForeground(new Color(0, 153, 51));

ver.setForeground(new Color(0, 153, 51));

cop.setForeground(new Color(0, 153, 51));

setSize(new Dimension(330, 215));

Dimension scr = Toolkit.getDefaultToolkit().getScreenSize();
```

```
        setLocation((scr.width - getSize().width) / 2, (scr.height - getSize().height)

                / 2);

}

public void actionPerformed(ActionEvent e) {

        if (e.getSource() == ok) {

                setVisible(false);

        }

}



}




/**

 * <p>Title:        All                    </p>

 * <p>Description: This class is responsible to provide vector element by type.

 *          </p>

 * <p>Copyright: Copyright (c) 2005</p>

 * <p>Company: Naval Postgraduate School, Monterey, CA</p>

 * @author Min Kim

 * @version 1.0
 */

import java.awt.*;
import java.awt.geom.*;

class All {
	int type;
	Color clr;
	Rectangle r;
	String s;
	Font f;
	Point loc;
	Image img;
public All(int type, Color clr, Rectangle r) {
	this.type = type;
	this.clr = clr;
	this.r = r;
}
public All(int type, Image img, Point loc) {
	this.type = type;
```

```
          this.img = img;

          this.loc = loc;

     }

}
```

LIST OF REFERENCES

Beutel, J. (1999). ***Geolocation in a picoradio environment,*** M.S. thesis, ETH Zurich, Electronics Lab.

Cerpa, A., Elson, J., Estrin, D., Girod, L., Hamilton, M. & Zhao, J. (2001) Habitat monitoring: application driver for wireless communications technology. ***ACM SIGCOMM Computer Communication Review,*** 31(2), 20–41.

Clouqueur, T., Phipatanasuphorn, V., Ramanathan, P., & Saluja, K.K. (2002, September). Sensor deployment strategy for target detection. ***ACM WSNA,*** 42–48.

Conover, J. (1999, November). Policy-based network management. ***Network Computing***.

Dhillon, S.S., Chakrabarty, K., & Iyengar, S.S. (2002). Sensor placement for grid coverage under imprecise detections. ***Int. Conf. inf. Fusion (FUSION),*** 2, 1581-1587.

Goel, S. & Imieli, T. (2001). Prediction-based monitoring in sensor networks: taking lessons from mpeg. ***Technical report,*** Rutgers University.

Google Earth (2005). [http://earth.google.com/]. (last accessed 11/05).

Hollar, S.E.-A. (2000). ***Cots dust.*** Master's thesis, University of California, Berkeley.

Howard, A., Mataric, M.J., & Sukhatme, G.S. (2002, June). Mobile sensor network deployment using potential fields: a distributed, scalable, solution to the area coverage problem, ***6th Int. Symp. Distributed Autonomous Robotics Syst,*** 299–308.

Iranle, A., Maleki, M., &Pedram, M. (2005). Energy efficient strategies for deployment of a two-level WSN. ***Proecedigns of the 2005 international symposium on Low power electronics and design,*** 233-238.

International Telecommunication Union(ITU). (1992). CCITT recommendation X. 700. ***Management framework for open systems interconnection (OSI) for CCITT application.***

International Telecommunication Union(ITU). (1996, May). ITU-T M. 3010. ***Principles for a telecommunications management network.***

Karl, H., & Willig, A. (2005). ***Protocols and Architectures for WSNs.*** England: John Wiley & sons Ltd.

Mainwaring, A.,Polastre, J., Szewczyk, R., Culler, D., & Anderson, J.(2002, September). WSNs for Habitat Monitoring. ***Porceedings of the 1st ACM Workshop on WSNs and Application.***

Meguerdichian, S., Koushanfar, F., Potkonjak, M., & Srivastava, M.B. (2001, April). Coverage problems in wireless ad-hoc sensor networks. *IEEE INFOCOM*, 3, 1380-1387.

Mehrotra, S. (2001, July). ***Distributed algorithms for tasking large sensor network.*** Thesis submitted to the faculty of Virginia Polytechnic Institute and State University.

Ruiz, L. B., Nogueira, J.M.s., & Loureiro, A.A.F. (2003, Febrary). MANNA: a management architecture for WSNs. ***IEEE Commun. Mag.,*** 41(2), 116-125.

Ruiz, L.B.,Siqueira,I.G., Oliveira, L.B., Wong, H.C, Nogueira, J.M.S., & Loureiro, A.A.F. (2004.October). Sensor networks: Fault management in event-driven WSNs. ***Proceedings of the 7th ACM international symposium on modeling, analysis and simulation of wireless and mobile systems.***

Ruiz, L. B., Nogueira, J.M.S., & Loureiro, A.A.F. (2005). Sensor Network Management. In M. Ilayas & I. Mahgoub (Eds, chap. 3) ***Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems.*** Boca Raton, FL: CRC Press LLC.

Savvides, A., Han, C.C., & Srivastava, M. (2001, August) Dynamic fine-grained localization in adhoc networks of sensors, ***Proc. ACM MobiCom'01.***

Savvides, A., Park, S., & Srivastava., M. (2001, June) On modeling networks of wireless microsensors. ***Joint Int. Conf. Measurement Modeling Computer Syst.***, 318-319, Cambridge, MA.

Savarese, C., Rabaey, J., & Beutel, J. (2001, May). Locationing in distributed ad-hoc WSNs, Proc. IEEE ICASSP'01. Matrix Pencil for Positioning in Wireless Ad Hoc Sensor Network 27

Tilac, S., Abu-Ghazaleh, N.B., & Heinzelman, W. (2002, September). Infrastructure trade-offs for sensor networks. ***ACM WSNA'02,*** 49-58

Vieira, M.A., Vieira, L.F., Ruiz, L.B., Loureiro, A.A. & Fernandes, A.O. (2003, October). Scheduling nodes in WSN: a Voronoi approach. ***IEEE LCN – Local Computer Network.***

Wang, Q., Hassanein, H., & Xu, K. (2005). A practical perspective on WSNs. In M. Ilayas & I. Mahgoub (Eds, chap. 9) ***Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems.*** Boca Raton, FL: CRC Press LLC.

Willing, A., Shah, R., Rabaey, J., & Wolisz, A. (2002, August). Altruists in the PicoRadio sensor network. ***4th IEEE Int. Workshop Factory Commun. Syst.,*** 175-184.

Wu, K. & Harms, J. (2001, November). QoS support in mobile ad hoc networks. ***Crossing Boundaries – GSA J. Univ. Alberta,*** 1(1), 92–106.

Zou, Y., & Chakrabarty, K. (2003, March). Sensor deployment and target localization based on virtual forces. ***IEEE INFOCOM,*** 2, 1293-1303.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Gurnider Singh
   Naval Postgraduate School
   Monterey, California 4.

4. Arijit Das
   Naval Postgraduate School
   Monterey, California

5. Min Young Kim
   Korean National Defense University
   Republic of Korea